UIUCDCS-R-77-896

UILU-ENG 77 1750

DESIGN OF IRREDUNDANT MULTIPLE-LEVEL MOS NETWORKS FOR MULTIPLE-OUTPUT AND INCOMPLETELY SPECIFIED FUNCTIONS

by

Chi-Chung Yeh

September 1977

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-77-896

DESIGN OF IRREDUNDANT MULTIPLE-LEVEL MOS NETWORKS FOR MULTIPLE-OUTPUT AND INCOMPLETELY SPECIFIED FUNCTIONS

by

Chi-Chung Yeh

September 1977

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois

This work was supported in part by the National Science Foundation under Grant No. MCS77-09744 and was submitted in partial fulfillment for the degree of Master of Science in Computer Science, 1977.

ACKNOWLEDGEMENT

The author is greatly grateful to Professor Saburo Muroga for his enthusiastic guidance and encouragement during the preparation of this thesis, and particularly for his careful reading and valuable suggestions of the original manuscript.

The author also wishes to thank K. C. Hu for his helpful guidance and discussion.

TABLE OF CONTENTS

1. INTRODUCTION

The recent progress in integrated circuit technology has been improving the main drawback of MOS devices, slow speed, and the MOS logic circuits have become one of the most important logic families for digital systems. MOS logic circuits have many advantages over bipolar devices such as high packing density, lower power consumption, simple production processes, etc.

Since an MOS cell can theoretically realize an arbitrary negative function, it is generally referred to as a negative gate. Several algorithms for the synthesis of negative gate networks have been developed. An algorithm for designing two-level networks with a minimal number of negative gates was first derived by T. Ibaraki and S. Muroga [1]. The synthesis of multi-level networks with minimum numbers of negative gates was introduced by T. Nakamura, N. Tokura, and T. Kasami [2] based on a general structure of a feed-forward negative gate network. A similar approach to design multi-level or two-level networks with a minimal number of negative gates was independently devised by T. K. Liu [3]. Although these algorithms do guarantee the minimality of the number of negative gates, they consider neither the number of connections nor the complexity in each negative gate. Thus a network designed by the above algorithms may contain redundant connections or driver MOSFET's. Recently, a new algorithm named DIMN (Design of Irredundant MOS Network), which finds irredundant multi-level MOS networks with a minimum number of negative gates for a given set of functions, was introduced by H. C. Lai [4]. An MOS network is irredundant if any driver MOSFET in the network is removed (by short-circuiting or open-

circuiting the drain and source terminals of that MOSFET, as we dicuss later), then the network will not realize the original network output function any more. Lai's algorithm for designing irredundant multi-level MOS networks with a minimum number of negative gates contains Algorithm DIMN and Modified Algorithm DIMN. Algorithm DIMN (i.e., the simplest case of Lai's algorithm) is to design irredundant MOS networks with minimum number of negative gates for a completely specified single-output function. This Algorithm DIMN is modified for designing an irredundant MOS network for a set of completely specified output functions and is referred to as the Modified Algorithm DIMN. Algorithm DIMN and Modified Algorithm DIMN can easily be extended to the cases of an incompletely specified single-output function and a set of incompletely specified output functions, respectively. For multiple-output functions,two different MOS network configurations can be obtained by Lai's algorithm, that is, the networks with all the output functions restricted to the last level and the networks with all the output functions restricted to the last m levels, where m is the number of output functions.

Lai's algorithm except the case where the output functions restricted to the last level was implemented with a FORTRAN program by K. Yamamoto [5]. This paper describes a newly developed program which combines the implementation of the algorithm for realizing the networks with all the output functions restricted to the last level with Yamamoto's program. This program which is named program DIMN covers all the algorithms introduced in Lai's paper and has the capability to realize networks for any single-output or multiple-output, completely or incompletely specified function. The contents of this thesis are introduced below.

Chapter 2 reviews Lai's algorithm (Algorithm DIMN and Modified Algorithm DIMN). Chapter 3 discussed Yamamoto's program as well as the modifications of Yamamoto's program in detail. Chapter 4 explains how to prepare the input format for using the program and the restriction on the problem size that program DIMN can handle. Chapter 5 describes the output format of program DIMN. It also compares the results obtained by program DIMN under the conditions that the output functions are realized at the last level, with the results under the conditions that the output functions are realized at the last m levels. Finally, the networks obtained for several three and four variable functions and a complete listing of FORTRAN program DIMN are given in Appendices A and B, respectively.

2. THEORETICAL BACKGROUND

The power consumed in a static MOS network is approximately proportional to the number of MOS cells that the network has. Furthermore, an MOS network with fewer MOS cells tends to contain fewer connections or intercell connections than a network which has more cells. Therefore, the minimization of the number of MOS cells in MOS networks appears to be most important objective in designing most compact MOS networks [6].

The methods for designing negative gate networks with minimum number of negative gates which have been developed before Lai contain the following three steps.

Step 1: Find the minimum number of MOS cells required to realize the given functions.

Step 2: Obtain a function for each MOS cell.

Step 3: Design internal MOS cell configuration for each function obtained in Step 2.

Since the design of each cell is done in Step 3 separately from Step 1 and 2 (i.e., there is no interplay between these steps), each cell may become unnecessarily complex and the network designed may contain many redundant connections (MOSFET's). The aim of Lai's algorithm is to design an irredundant MOS network with a minimum number of cells. Instead of applying the above steps once, Lai's algorithm applies Step 2 and Step 3 iteratively. In Step 2 a function is obtained such that it contains as many don't care's as possible, then in Step 3, an irredundant MOS cell configuration is designed by utilizing these don't care's to the maximum extent.

In order to facilitate a review of Lai's algorithm, let us introduce the following definitions and terminologies.

Negative Function

A negative function is a switching function which has a disjunctive form consisting of complemented variables only. For example, function $f_1 = \bar{x}_1\bar{x}_3 \vee \bar{x}_1\bar{x}_2$ is a negative function and function $f_2 = x_1x_2\bar{x}_3 \vee \bar{x}_1x_3$ is not a negative function.

Negative Gate

A negative gate is a logic gate which can realize a negative function.

N-Cube

This is a lattice which represents the switching functions with n external input variables. There are 2^n vertices in an N-cube and each vertex corresponds to an input vector of the functions. Vertices with same weight (number of ones in an input vector assigned to the vertex) are in the same level. Every pair of vertices is connected by an edge if their corresponding input vectors differ in one bit position only.

Labeled N-Cube

An N-cube is referred to as a labeled N-cube with respect to functions $f_1,\ldots,f_m$ when a binary integer,

$$L(A) \equiv \ell(A;f_1,\ldots,f_m) = \sum_{i=1}^{m} f_i(A)\, 2^{m-i}$$

is attached to each vertex A of N-cube as a label. An example of a labeled

3-cube for one bit full adder ($N = 3$, f_1 = carry = $x_1x_2 \vee x_1x_3 \vee x_2x_3$, f_2 = sum = $\bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2x_3$) is shown in Fig. 2.1.

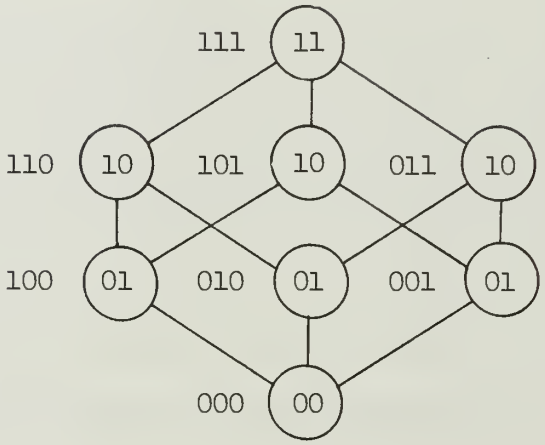


Fig. 2.1 Labeled 3-cube for one bit full adder ($f_1 = c = x_1x_2 \vee x_1x_3 \vee x_2x_3$, $f_2 = s = \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2x_3 \vee x_1x_2x_3$).

Directed Edge

When two vectors $A = (a_1,\ldots,a_n)$ and $B = (b_1,\ldots,b_n)$ are given such that $a_i \geq b_i$ for every i, this is denoted with $A \geq B$. If $a_i > b_i$ furthermore holds with some i, this is denoted with $A > B$. The edge between vertices A and B in an N - cube is directed from A to B if $A > B$ and is denoted by $\overrightarrow{A\,B}$.

Inverse Edge

A directed edge $\overrightarrow{A\,B}$ is said to be an inverse edge of a labeled N-cube if and only if

$$\ell(A;f_1,\ldots,f_m) > \ell(B;f_1,\ldots,f_m)$$

For example, every directed edge in Fig. 2.1 is an inverse edge.

A generalized form of a loop-free network consisting of R negative gates is shown in Fig. 2.2, where R is the number of negative gates (MOS cells) required to realize the function f, $x_1,\ldots,x_n$ are n external input variables, and g_i is the i-th gate counting from the left for i = 1,...,R with g_R being the output gate. Let $u_i(x_1,\ldots,x_n)$ be the function realized by gate g_i with respect to the external input variables $x_i,\ldots,x_n$. The values of u_i for combinations of inputs $x_1,\ldots,x_n$ will be called the components of u_i. Since every gate is a negative gate, u_i is negative with respect to the inputs $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$ of g_i and is an incompletely specified negative function of n + i - 1 variables. Our design objective is to find a negative gate network with a minimum number of negative gates. When the number of gates is minimized, R will be referred to as R_f.

Negative Function Sequence

In the generalized form of the loop-free network in Fig. 2.2, the sequence of ordered negative functions $u_1,\ldots,u_R$ is called a negative function sequence of length R for function f and is denoted by NFS(R,f) = $(u_1,\ldots,u_R)$. Here, u_R represents the output of the network, i.e., u_R = f.

Partially Specified Negative Function Sequence

A partially specified negative function sequence of length R and degree i for a function f is a sequence of R negative functions denoted by $NFS^i(R,f) = (u_1,\ldots,u_i,u^*_{i+1},\ldots,u^*_{R-1},u_R)$ such that $u^*_{i+1},\ldots,u^*_{R-1}$ are unspecified functions (i.e., all components of $u^*_{i+1},\ldots,u^*_{R-1}$ are don't

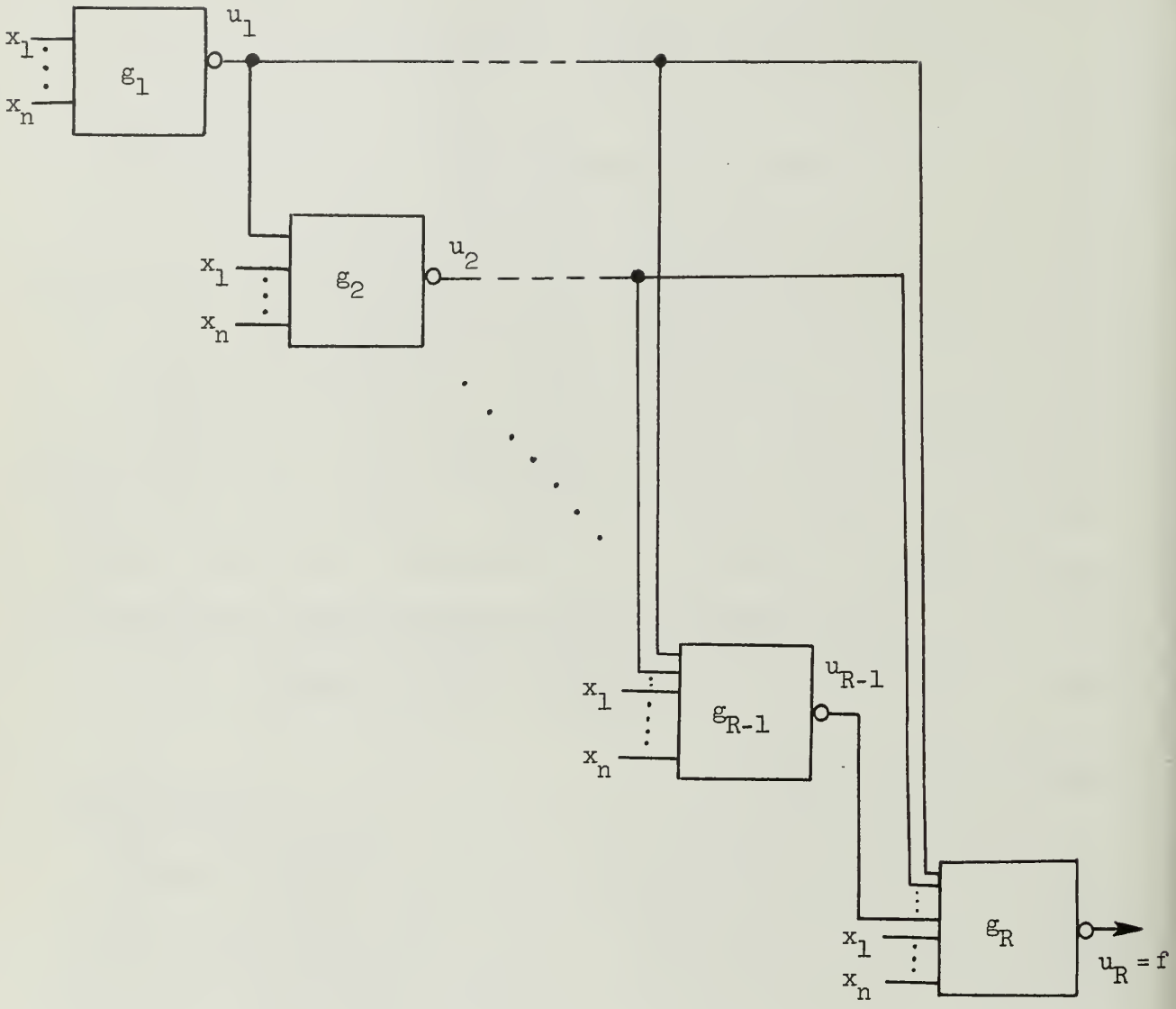


Fig. 2.2 Generalized form of a loop-free network consisting of R negative gates.

cares) and $u_1,\ldots,u_i$ are completely specified functions of $x_1,\ldots,x_N$. Here, $(u_1,\ldots,u_i)$ is an NFS of length i. In particular, $NFS^o(R,f) = (u_1^*,\ldots,u_{R-1}^*,u_R)$ where superscript o means that no function except $u_R = f$ is specified and all components of u_i^* are don't cares.

An NFS(R,f) can be obtained by assigning 0's and 1's to those don't cares of a feasible partially specified $NFS^i(R,f)$. This is called a completion of $NFS^i(R,f)$. A completion of $NFS^i(R,f)$ is called feasible if the resulting labeled N-cube has no inverse edge. A partially specified $NFS^i(R,f)$ is feasible if there exists at least one feasible completion of $NFS^i(R,f)$; otherwise $NFS^i(R,f)$ is infeasible.

Algorithm CMNL: (Conditional Minimum Labeling)

This algorithm obtains a conditional minimum labeling, $\underline{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},\underline{u_i},\ldots,\underline{u_{R_f-1}},f)$, based on a given feasible $NFS^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},u_i^*,\ldots,u_{R_f-1}^*,f)$ for a function f. (Notice the underline.) The $\underline{NFS}^{i-1}(R_f,f)$ is a completion of $NFS^{i-1}(R_f,f)$ such that the label $\ell(A;u_1,\ldots,u_{i-1},\underline{u_i},\ldots,\underline{u_{R_f-1}},f) \equiv L_{mn}^{f,i-1}(A)$ assigned for each vertex A in the corresponding N-cube has the minimum value among all feasible completions of $NFS^{i-1}(R_f,f)$.

Step 1 Assign as $L_{mn}^{f,i-1}(I)$ the value $\sum_{k=1}^{i-1} 2^{R_f-k} u_k(I) + f(I)$, where I represents the top vertex in the N-cube.

Step 2 When $L_{mn}^{f,i-1}(A)$ is assigned to each vertex A of weight w ($1 \le w \le n$) in the N-cube, assign as $L_{mn}^{f,i-1}(B)$ to each vertex B of weight w - 1 the smallest binary integer satisfying the following three conditions:

(a) The k-th most significant bit of $L_{mn}^{f,i-1}(B)$ is $u_k(B)$ for $k = 1,\ldots,i-1$;

(b) The least significant bit of $L_{mn}^{f,i-1}(B)$ is $f(B)$;

(c) $L_{mn}^{f,i-1}(B) \geq L_{mn}^{f,i-1}(A)$ for every directed edge $\overrightarrow{A\,B}$ terminating at B.

Step 3 Repeat Step 2 until $L_{mn}^{f,i-1}(0)$ is assigned, where 0 denotes the bottom vertex in the corresponding N-cube.

Step 4 The k-th most significant bit of $L_{mn}^{f,i-1}(A)$ is denoted by $u_k(A)$ for $k = i,\ldots,R_f-1$ and the completion of $NFS^{i-1}(R_f,1)$, $\underline{NFS}^{i-1}(R_f,f) = (u_i,\ldots,u_{i-1},\underline{u}_i,\ldots,\underline{u}_{R_f-1},f)$, has been obtained.

Algorithm CMXL: (Conditional Maximum Labeling)

This algorithm obtains a conditional maximum labeling, $\hat{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},\hat{u}_i,\ldots,\hat{u}_{R_f-1},f)$, for a given feasible $NFS^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},u_i^*,\ldots,u_{R_f-1}^*,f)$. The $\hat{NFS}^{i-1}(R_f,f)$ is a complete specification of $NFS^{i-1}(R_f,f)$ such that the label $\ell(A;u_1,\ldots,u_{i-1},\hat{u}_i,\ldots,\hat{u}_{R_f-1},f) \equiv L_{mx}^{f,i-1}(A)$ assigned to each vertex A in the corresponding N-cube takes the maximum value among all feasible completions of $NFS^{i-1}(R_f,f)$.

Step 1 Assign as $L_{mx}^{f,i-1}(0)$ the value $\sum_{k=1}^{i-1} 2^{R_f-k}\,u_k(0) + \sum_{k=i}^{R_f-1} 2^{R_f-k} + f(0)$

Step 2 When $L_{mx}^{f,i-1}(A)$ is assigned to each vertex A of weight w ($0 \leq w \leq n-1$) in N-cube, assign as $L_{mx}^{f,i-1}(B)$ to each vertex B of weight w+1 the largest binary integer satisfying the following three conditions:

(a) The k-th most significant bit of $L_{mx}^{f,i-1}(B)$ is $u_k(B)$ for $k = 1,\ldots,i-1$;

(b) The least significant bit of $L_{mx}^{f,i-1}(B)$ is f(B);

(c) $L_{mx}^{f,i-1}(B) \leq L_{mx}^{f,i-1}(A)$ for every directed edge $\overrightarrow{BA}$ originating from B.

Step 3 Repeat Step 2 until $L_{mx}^{f,i-1}(I)$ is assigned.

Step 4 The k-th most significant bit of $L_{mx}^{f,i-1}(A)$ is denoted by $\hat{u}_k(A)$ for $k = i,\ldots,R_f-1$ and the completion of $NFS^{i-1}(R_f,f)$, $\hat{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},\hat{u}_i,\ldots,\hat{u}_{R_f-1},f)$, has been obtained.

Algorithm DIMN: Design of irredundant MOS networks with a minimum number of MOS cells for a given function f. R_f represents the minimum number of MOS cells.

Step 1 Let $NFS^{o}(R_f,f) = (u_1^*,\ldots,u_{R_f-1}^*,f)$ and set i = 1. (If R_f is not known at this step, it will be obtained after $\underline{NFS}^{o}(R_f,f)$ is obtained by applying CMNL in Step 2).

Step 2 Use algorithm CMNL to obtain $\underline{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},\underline{u}_i,\ldots,\underline{u}_{R_f-1},f)$.

Step 3 Use algorithm CMXL to obtain $\hat{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},\hat{u}_i,\ldots,\hat{u}_{R_f-1},f)$.

Step 4 Obtain function $\tilde{u}_i$ by setting:

$\tilde{u}_i(A) = 0$, if $\underline{u}_i(A) = \hat{u}_i(A) = 0$;

$\tilde{u}_i(A) = 1$, if $\underline{u}_i(A) = \hat{u}_i(A) = 1$;

$\tilde{u}_i(A) = *$, if $\underline{u}_i(A) = 0$ and $\hat{u}_i(A) = 1$.

The function, $\tilde{u}_i$, obtained by this step is the MPF (Maximum

Permissible Function) for the feasible $NFS^{i-1}(R_f,f)$. After applying this step, the negative function sequence, $(u_1,\ldots,u_{i-1},\tilde{u}_i,u^*_{i+1},\ldots,u^*_{R_f-1},f)$ is denoted as $\tilde{NFS}^{i-1}(R_f,f)$.

Step 5 Obtain an irredundant configuration for MOS cell g_i with function $\tilde{u}_i$ with respect to $x_1,\ldots,x_n,u_1,\ldots,u_{i-1}$. (See Lai's thesis Chapter 5 for detail). Let u_i denote the function realized by this MOS cell (u_i is now a completion of $\tilde{u}_i$ with respect to $x_1,\ldots,x_n$). Thus $NFS^i(R_f,f) = (u_1,\ldots,u_i,u^*_{i+1},\ldots,u^*_{R_f-1},f)$ has been obtained.

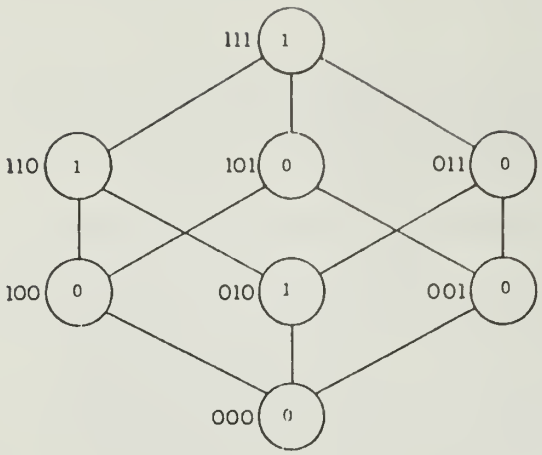
Step 6 If $i = R_f-1$, design an irredundant cell configuration for the output MOS cell with function f with respect to $x_1,\ldots,x_n,u_1,\ldots,u_{R_f-1}$ and terminate this algorithm; otherwise set $i = i + 1$ and go to Step 2.

It should be noted that this algorithm usually requires at most R_f-1 iterations, deriving $\underline{NFS}^{i-1}(R_f,f)$, $\hat{NFS}^{i-1}(R_f,f)$, $\tilde{NFS}^{i-1}(R_f,f)$ and $NFS^i(R_f,f)$ in each iteration. However, if $\tilde{NFS}^{i-1}(R_f,f)$ becomes a completely specified function with respect to $x_1,\ldots,x_N$, for some $i < R_f$(i.e., only one unique completion exists for this $NFS^{i-1}(R_f,f)$ and $\underline{NFS}^{i-1}(R_f,f) = \hat{NFS}^{i-1}(R_f,f)$ holds), then $\tilde{NFS}^{i-1}(R_f,f) = NFS^i(R_f,f) = NFS(R_f,f)$ must hold. In this case, the algorithm requires only i iterations, but only Step 5 of the algorithm must be executed additional $R_f - i - 1$ iterations without executing other steps in order to obtain the irredundant MOS cell configuration for $u_{i+1},\ldots,u_{R_f-1}$. This fact will help reducing the computation time when the algorithm is implemented in computer program.

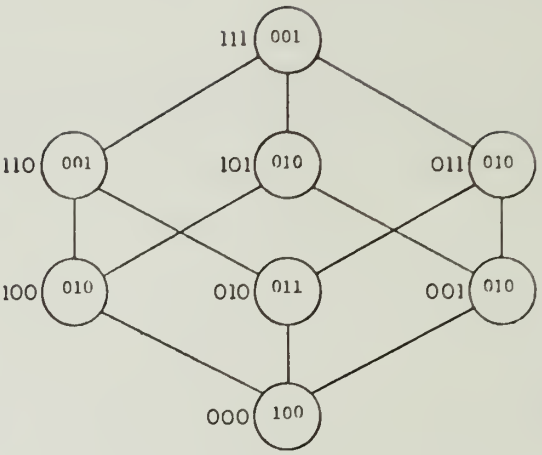
Step-by-step applications of Algorithm DIMN will be illustrated with an example in Fig. 2.3.

Example 2.1 The 3-cube for given f, i.e., $f = x_1x_2 \vee x_2\bar{x}_3$, is shown in Fig. 2.3(a). Step 2 and Step 3 of Algorithm DIMN obtain $\underline{NFS}^0(3,f)$ and $\hat{NFS}^0(3,f)$ as shown in (b) and (c), respectively. Step 4 then compares $\underline{u}_1$ and $\hat{u}_1$ and obtains $\tilde{u}_1$ and $\tilde{NFS}^0(3,f)$ as shown in (d). The $NFS^1(3,f)_1$ $(\bar{x}_1,u_2^*,f)$ in (e) is obtained by Step 5. Step 5 also obtains other two irredundant MOS cell configurations for $\tilde{u}_1$ and their corresponding $NFS^1(3,f)_2$ and $NFS^1(3,f)_3$ as shown in (j) and (o), respectively. After Step 6, the algorithm returns to Step 2 and obtains $\underline{NFS}^1(3,f)_1$, $\hat{NFS}^1(3,f)_1$, $\tilde{NFS}^1(3,f)_1$ and $NFS^2(3,f)_1$ as shown in (f), (g), (h), and (i), respectively. Since $R_f=3$, $NFS^2(3,f)_1$ is a completely specified negative function sequence with respect to x_1,x_2, and x_3. To finish the design, Step 6 of the algorithm obtains an irredundant MOS cell configuration for f. As mentioned above, Step 5 of the algorithm can also obtain $NFS^1(3,f)_2 = (\bar{x}_2,u_2^*,f)$ or $NFS^1(3,f)_3 = (\bar{x}_3,u_2^*,f)$ as shown in (j) and (o), respectively. From (j), the algorithm obtains $\underline{NFS}^1(3,f)_2$, $\hat{NFS}^1(3,f)_2$, $\tilde{NFS}^1(3,f)_2$, and $NFS^2(3,f)_2$ as shown in (k), (ℓ), (m), and (n), respectively. $NFS(3,f)_2$ and $NFS(3,f)_1$ are indistinguishable, since the permutation of u_1 and u_2 in $NFS(3,f)_2$ results in $NFS(3,f)_1$. The corresponding MOS network for both NFS(3,f)'s is shown in (s). Similarly, from (o), the algorithm obtains $\underline{NFS}^1(3,f)_3$, $\hat{NFS}^1(3,f)_3$, and $\tilde{NFS}^1(3,f)_3 = NFS(3,f)_3$, as shown in (p), (q), and (r), respectively. The MOS network corresponding to $NFS(3,f)_3$ is shown in (t).

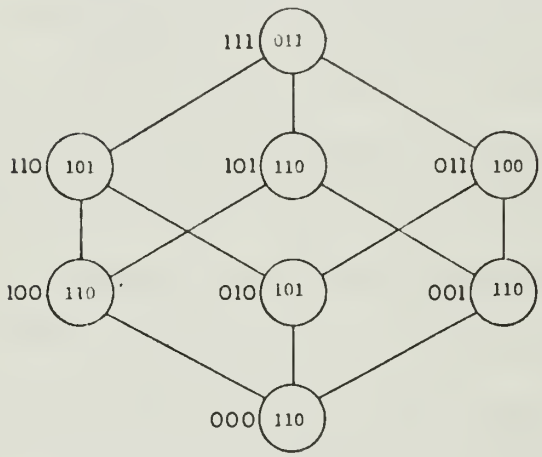
As observed from the above example, given a $NFS^{i-1}(R_f,f)$, sequences $\underline{NFS}^{i-1}(R_f,f)$, $\hat{NFS}^{i-1}(R_f,f)$, and $\tilde{NFS}^{i-1}(R_f,f)$ are uniquely determined in Step 2, 3, and 4 of Algorithm DIMN, respectively. On the other hand, Step 5 may obtain more than one irredundant MOS cell configuration for a given $\tilde{u}_i$.



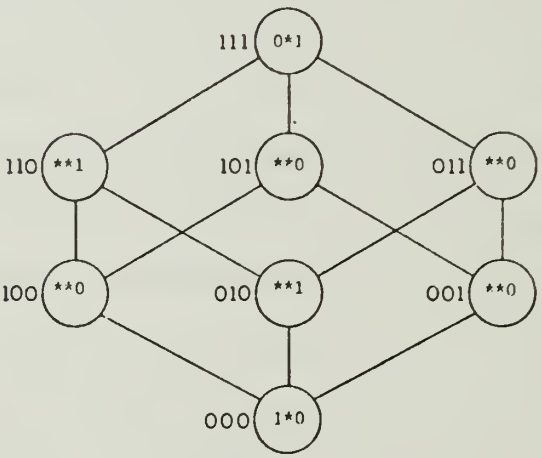
(a) $C_3(f)$ for $f = x_1x_2 \vee x_2\bar{x}_3$.



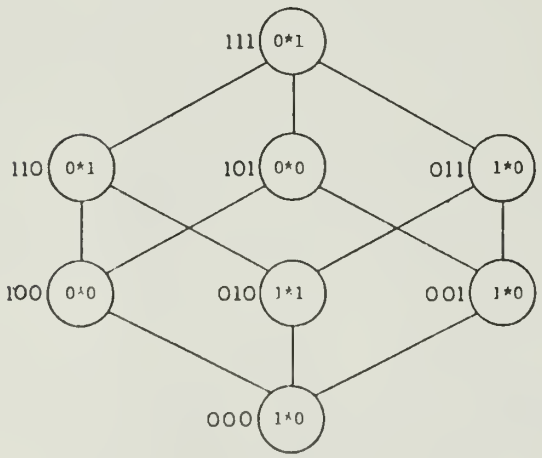
(b) $\underline{NFS}^0(3,f) = (\underline{u}_1, \underline{u}_2, f)$.



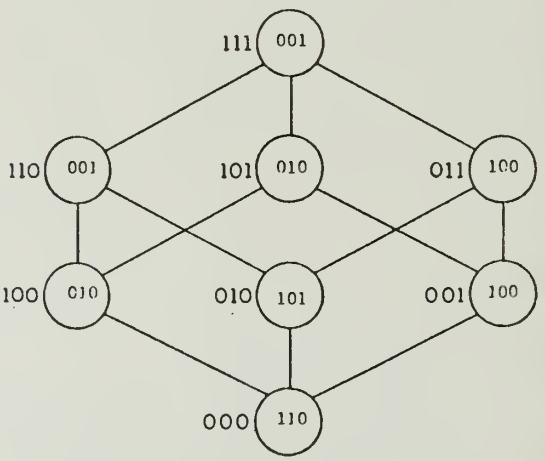
(c) $\hat{NFS}^0(3,f) = (\hat{u}_1, \hat{u}_2, f)$.



(d) $\tilde{NFS}^0(3,f) = (\tilde{u}_1, u_2^*, f)$.

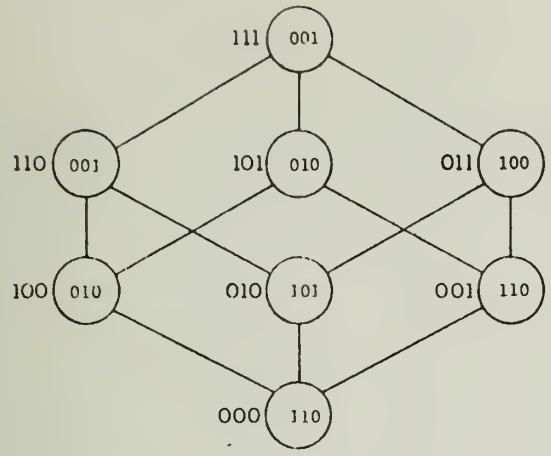


(d) $NFS^1(3,f)_1 = (\bar{x}_1, u_2^*, f)$.

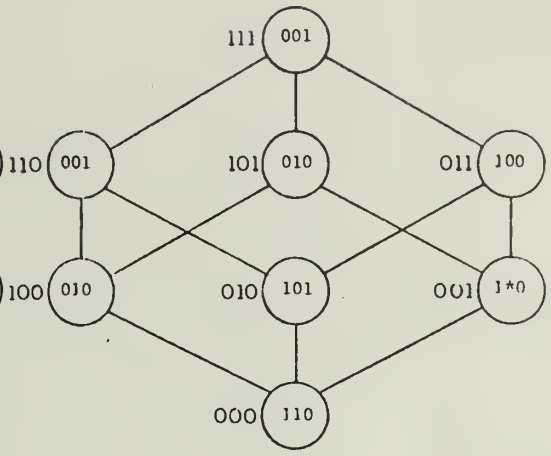


(f) $\underline{NFS}^1(3,f)_1 = (\bar{x}_1, \underline{u}_2, f)$.

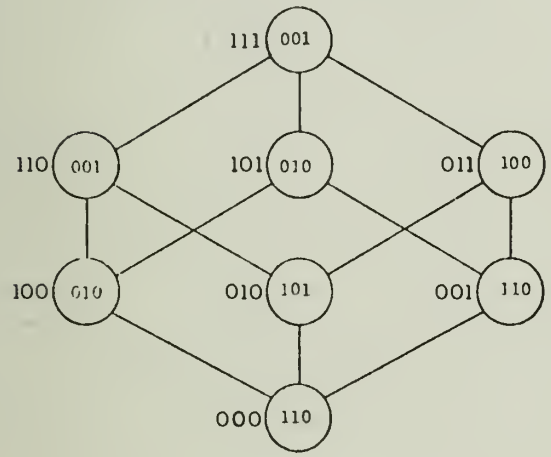
Fig. 2.3 Example 2.1.



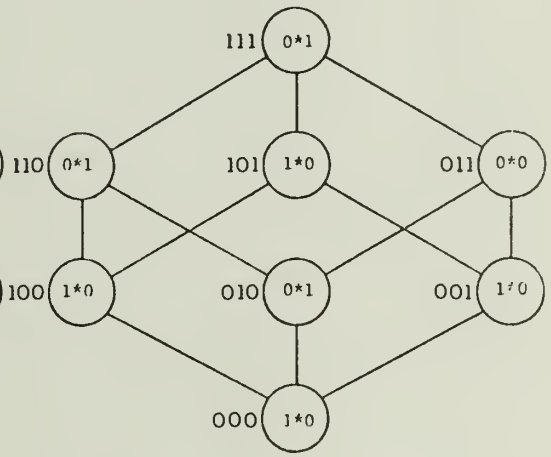
(g) $\widehat{NFS}^1(3,f)_1 = (\bar{x}_1, \hat{u}_2, f)$.



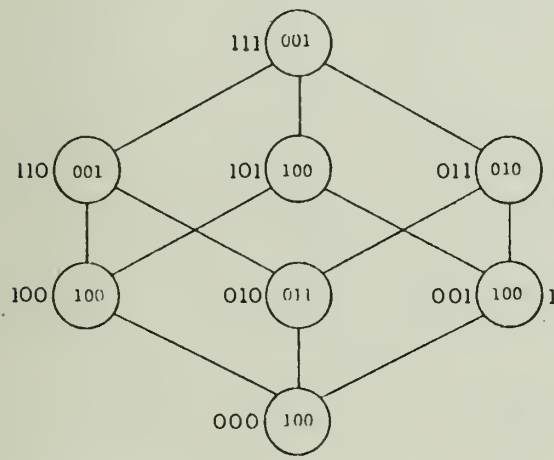
(h) $\tilde{\widehat{NFS}}^1(3,f)_1 = (\bar{x}_1, \tilde{\hat{u}}_2, f)$.



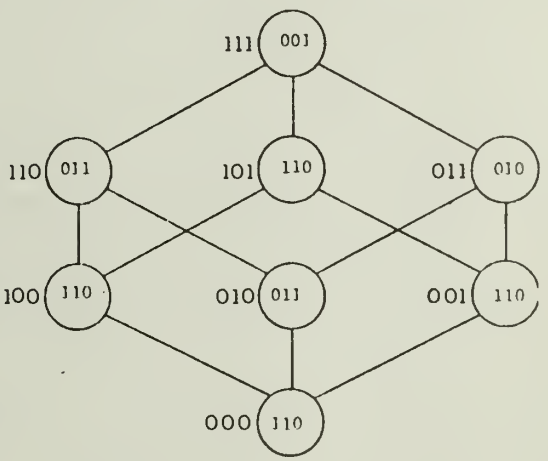
(i) $NFS^2(3,f)_1 = NFS(3,f)_1 = (\bar{x}_1, \bar{x}_2, f)$



(j) $NFS^1(3,f)_2 = (\bar{x}_2, u_2^*, f)$ obtained from (d).

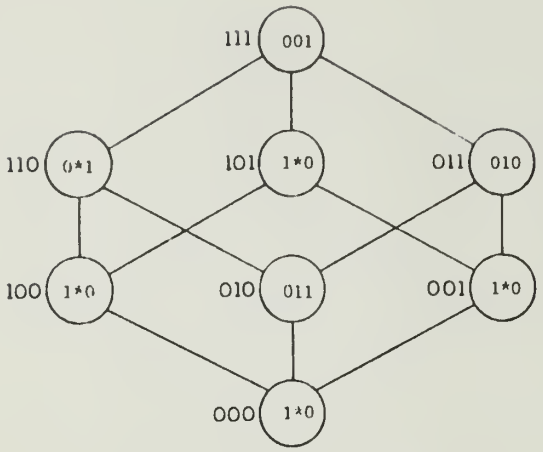


(k) $\underline{NFS}^1(3,f)_2 = (\bar{x}_2, \underline{u}_2, f)$.

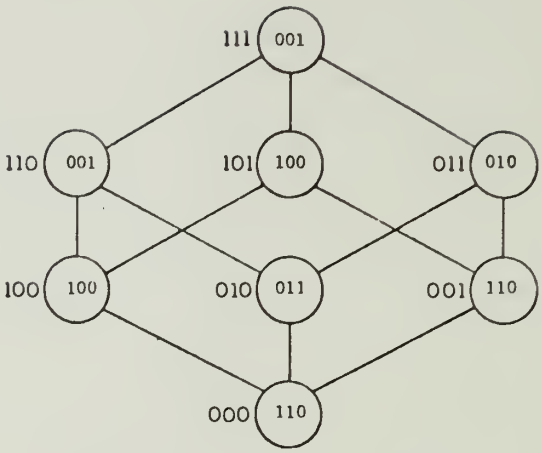


(l) $\widehat{NFS}^1(3,f)_2 = (\bar{x}_2, \hat{u}_2, f)$.

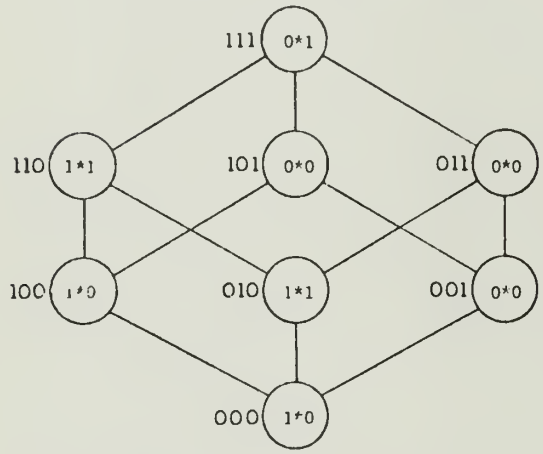
Fig. 2.3 (Continued)



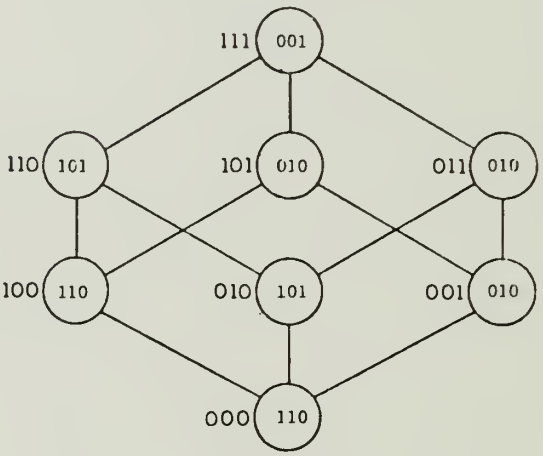
(m) $\hat{NFS}^1(3,f)_2 = (\bar{x}_2, \hat{u}_2, f)$.



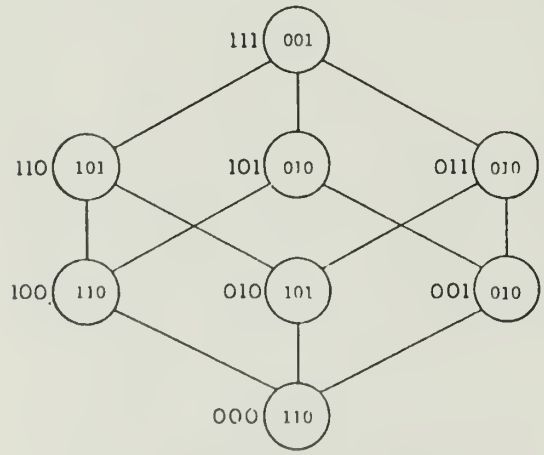
(n) $NFS^2(3,f)_2 = NFS(3,f)_2 = (\bar{x}_2, \bar{x}_1, f)$.



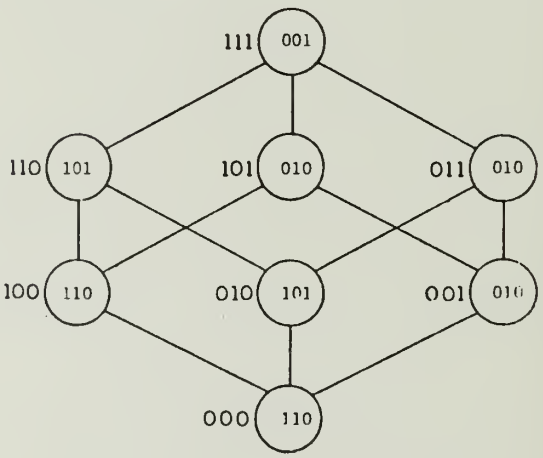
(o) $NFS^1(3,f)_3 = (\bar{x}_3, u_2^*, f)$ obtained from (d).



(p) $\underline{NFS}^1(3,f)_3 = (\bar{x}_3, \underline{u}_2, f)$.

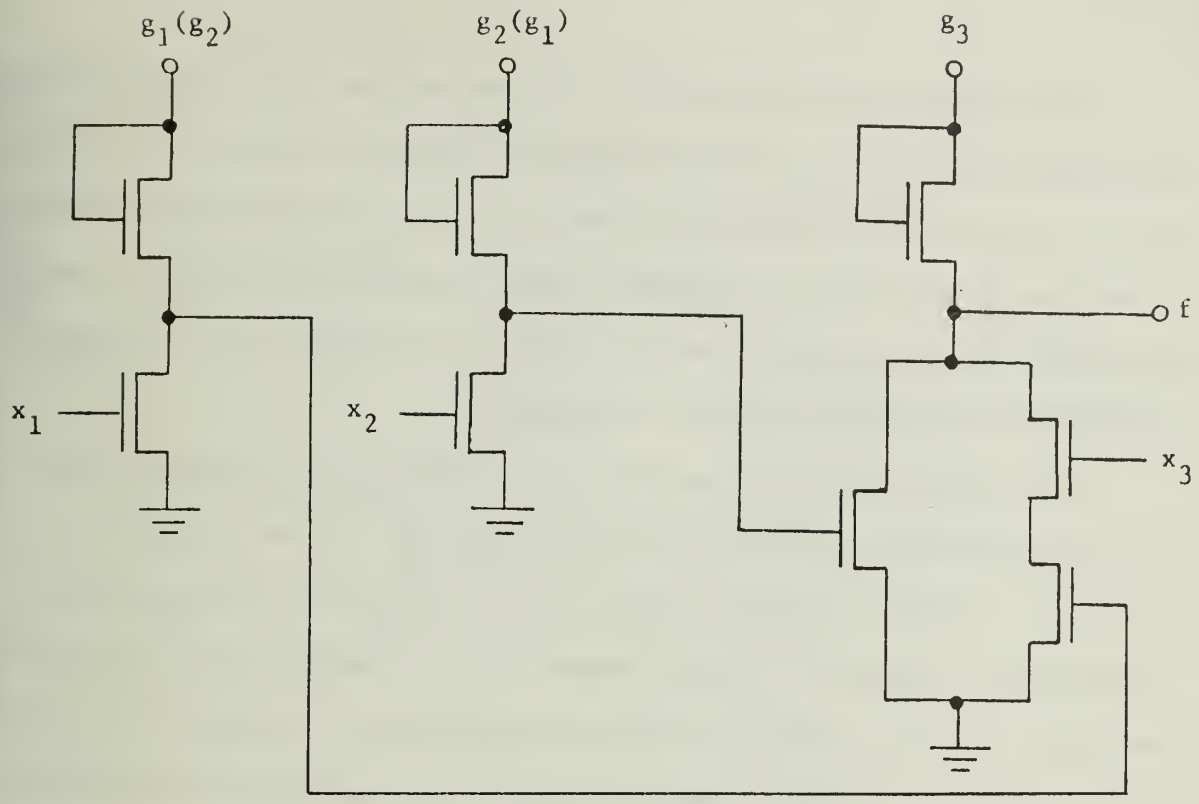


(q) $\hat{NFS}^1(3,f)_3 = (\bar{x}_3, \hat{u}_2, f)$.

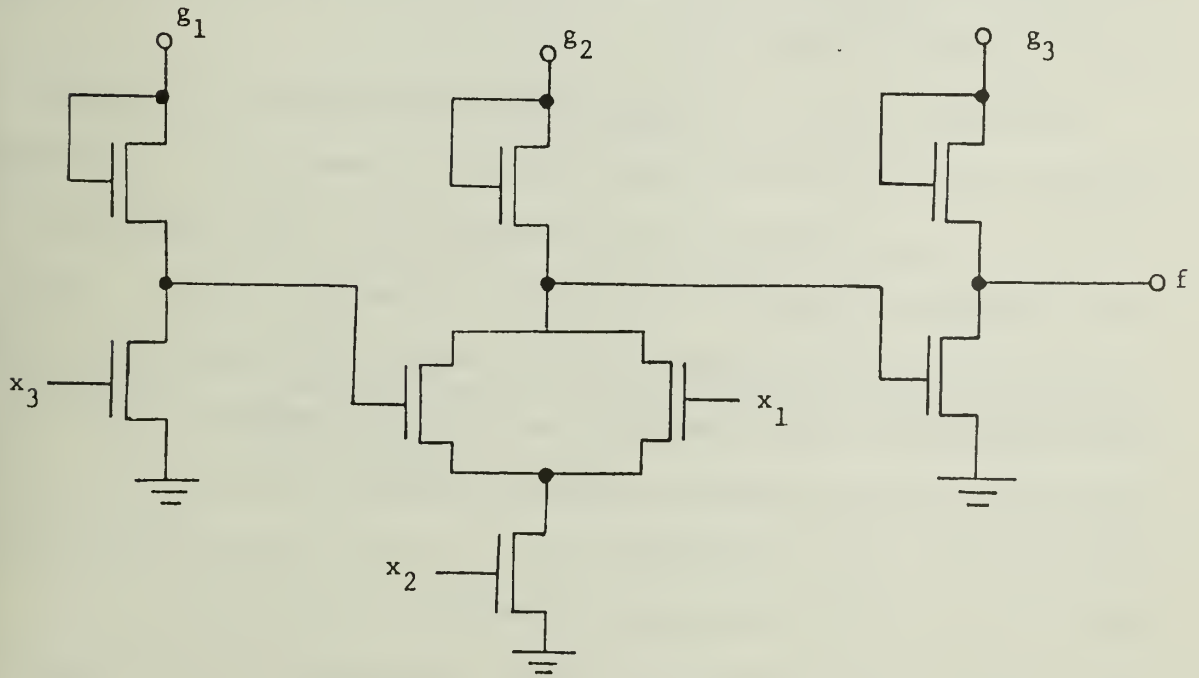


(r) $\tilde{NFS}^1(3,f)_3 = NFS(3,f) = (\bar{x}_3, \bar{f}, f)$.

Fig. 2.3 (Continued)



(s) Irredundant MOS network corresponding to NFS(3,f)'s in (i) and (n).



(t) Irredundant MOS network corresponding to $NFS(3,f)_3$ in (r).

Fig. 2.3 (Continued)

Another point to be noted is that the exhaustion of all alternatives in Step 5 may not give all possible irredundant MOS networks for the given function. If we are only interested in obtaining one irredundant MOS network for f, any one of $NFS^{i-1}(R_f,f)$ obtained in Step 5 of Algorithm DIMN is a solution. No matter which particular irredundant MOS cell configuration is chosen, the designed MOS network will be irredundant.

In the multiple-output function case, there are two algorithms for designing irredundant MOS networks for a given set of functions. One algorithm is for the case where given set of functions are restricted to be realized at the last m levels, where m is the number of output functions; the other algorithm is for the case where all output functions are restricted to be realized at the last level (in other words, any negative gate which realizes an output function is not allowed to feed other negative gates in the network). Only the latter case will be discussed below since the former case is discussed in [5].

The generalized form of a network in which functions $f_1,\ldots,f_m$ are realized at the last level is shown in Fig. 2.4, where $R = r + m$ and negative gates $g_{r+1},\ldots,g_{r+m}$ realize the given functions $f_1,\ldots,f_m$, respectively. Let us consider the subnetwork consisting of gates $g_1,\ldots,g_r,g_{r+1}$. This subnetwork is in the generalized form of networks consisting of $r+1$ negative gates shown in Fig. 2.2. Therefore, the sequence of functions $(u_1,\ldots,u_r,f_1)$ constitutes a negative function sequence of length $r+1$, i.e., $NFS(r+1,f_1)$. Similarly, the sequence of functions $(u_1,\ldots,u_r,f_i)$ is also a $NFS(r+1,f_i)$, for $i = 2,\ldots,m$. For convenience, let $NFS(r;f_1,\ldots,f_m) = (u_1,\ldots,u_r;$ $f_1,\ldots,f_m)$ be a negative function sequence corresponding to the network in

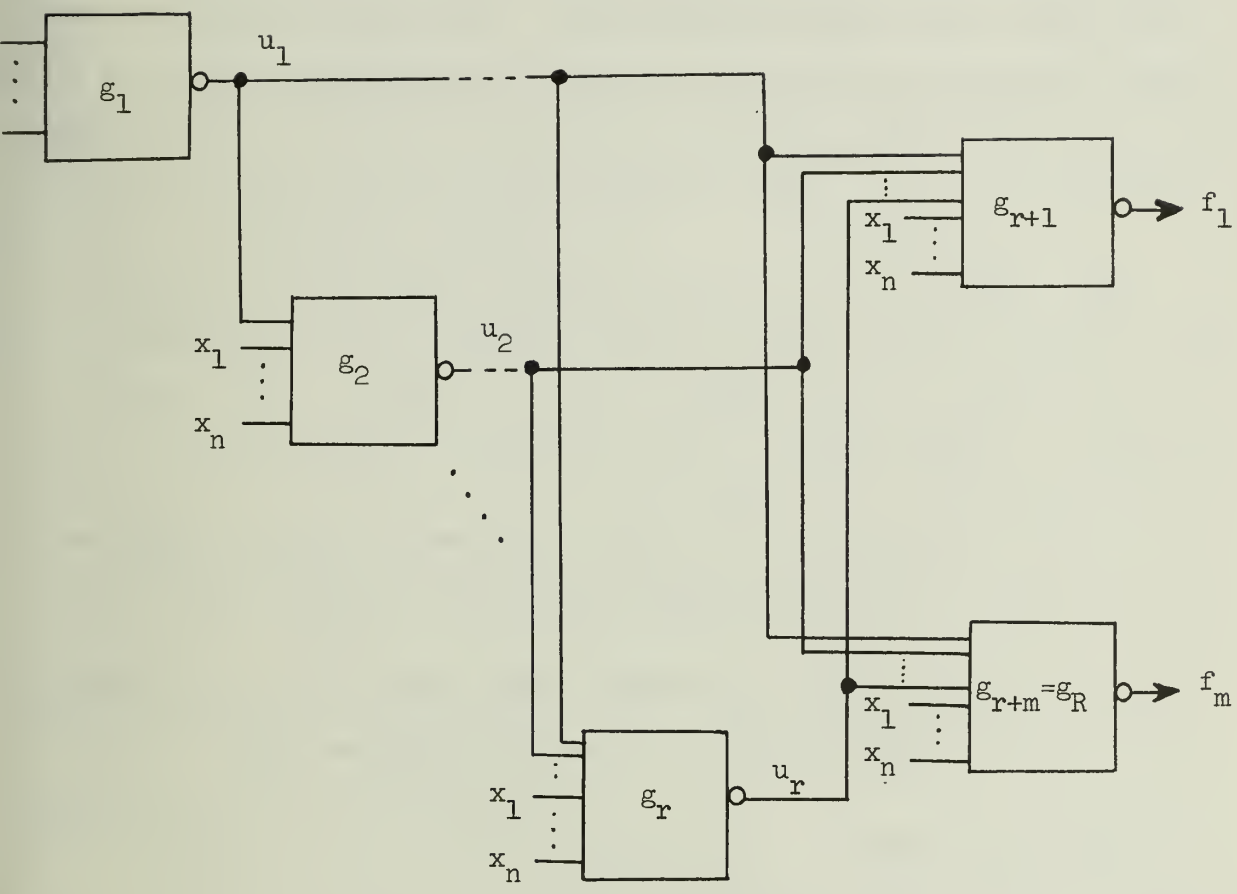


Fig. 2.4 Generalized form of a network which consists of R = r+m negative gates with all m output gates realized in the output level.

the form of Fig. 2.4 and $\ell(A;u_1,\ldots,u_r;f_1,\ldots,f_m)$ be the label assigned to each vertex A in the N-cube. It should be noted that the label consists of two parts: $(u_1,\ldots,u_r)$ and $(f_1,\ldots,f_m)$. $(u_1,\ldots,u_r)$ is considered as a binary interger of r bits and $(f_1,\ldots,f_m)$ is considered as a vector of m bits. The relationships between two labels in vertices A and B are defined below.

(1) $\ell(A;u_1,\ldots,u_r;f_1,\ldots,f_m) = \ell(B;u_1,\ldots,u_r;f_1,\ldots,f_m)$ if and only if $u_i(A) = u_i(B)$ for $i = 1,\ldots,r$ and $f_i(A) = f_i(B)$ for $i = 1,\ldots,m$;

(2) $\ell(A;u_1,\ldots,u_r;f_1,\ldots,f_m) > \ell(B;u_1,\ldots,u_r;f_1,\ldots,f_m)$ if and only if either $\ell(A;u_1,\ldots,u_r) > \ell(B;u_1,\ldots,u_r)$[†], or $\ell(A;u_1,\ldots,u_r) = \ell(B;u_1,\ldots,u_r)$ and $(f_1(A),\ldots,f_m(A)) > (f_1(B),\ldots,f_m(B))$.[‡]

The Algorithm CMNL can be modified as follows in order to design an irredundant MOS network in the form of Fig. 2.4. Let $L_{mn}^{f_1,\ldots,f_m;i-1}(A) = \ell(A;u_1,\ldots,u_{i-1},\underline{u}_i,\ldots,u_r)$ be the first part of the label to be assigned to each vertex A in the N-cube by the Modified Algorithm CMNL.

Modified Algorithm CMNL

Step 1 Assign vertex I the value

$$\sum_{k=1}^{i-1} 2^{R-k}\, u_k(I) + \sum_{j=1}^{m} 2^{m-j}\, f_j(I).$$

---

† $\ell(A;u_1,\ldots,u_r)$ represents the binary value of $(u_1,\ldots,u_r)$ of vertex A. Similarly with $\ell(B;u_1,\ldots,u_r)$.

‡ $(f_1(A),\ldots,f_m(A)) > (f_1(B),\ldots,f_m(B))$ means the vector comparison defined in page 6.

Step 2 When $L_{mn}^{f_1,\ldots,f_m;i-1}(A)$ is assigned to each vertex A of weight w ($1 \leq w \leq n$) in the N-cube, assign as $L_{mn}^{f_1,\ldots,f_m;i-1}(B)$ to each vertex B of weight w - 1 the smallest binary integer satisfying the following conditions for each directed edge $\overrightarrow{AB}$ in the N-cube.

(a) The k-th most significant bit of $L_{mn}^{f_1,\ldots,f_m;i-1}(B)$ is $u_k(B)$ for $k = 1,\ldots,i-1$;

(b) The m least significant bits of $\ell(B;u_1,\ldots,u_r;f_1,\ldots,f_m)$ are the values of the given output functions $f_1(B),\ldots,f_m(B)$.

(c) $L_{mn}^{f_1,\ldots,f_m;i-1}(B) \geq L_{mn}^{f_1,\ldots,f_m;i-1}(A)$ if $(f_1(A),\ldots,f_m(A)) \leq (f_1(B),\ldots,f_m(B))$; or $L_{mn}^{f_1,\ldots,f_m;i-1}(B) > L_{mn}^{f_1,\ldots,f_m;i-1}(A)$ if $(f_1(A),\ldots,f_m(A)) \not\leq (f_1(B),\ldots,f_m(B))$.†

Step 3 Repeat Step 2 until $L_{mn}^{f_1,\ldots,f_m;i-1}(0)$ is assigned.

Step 4 The k-th most significant bit of $L_{mn}^{f_1,\ldots,f_m;i-1}(A)$ is denoted by $\underline{u}_k(A)$ for $i = 1,\ldots,r$ and $\underline{NFS}^{i-1}(r+m;f_1,\ldots,f_m) = (u_1,\ldots,u_{i-1},\underline{u}_i,\ldots,\underline{u}_r;f_1,\ldots,f_m)$ has been obtained.

---

† $(f_1(A),\ldots,f_m(A)) \not\leq (f_1(B),\ldots,f_m(B))$ means that either $(f_1(A),\ldots,f_m(A)) > (f_1(B),\ldots,f_m(B))$ or they are incomparable, where two vectors $(f_1(A),\ldots,f_m(A))$ and $(f_1(B),\ldots,f_m(B))$ are "incomparable" if there exist two integers i and j, $1 \leq i, j \leq m$, such that $f_i(A) > f_i(B)$ and $f_j(A) < f_j(B)$.

Algorithm CMXL can be similarly modified and the modified algorithm CMXL assigns a maximum possible label $L_{mx}^{f_1,\ldots,f_m;i-1}(A) = \ell(A;u_1,\ldots,u_{i-1},\hat{u}_i,\ldots,\hat{u}_r)$ to each vertex A in the N-cube satisfying the same conditions as those in modified Algorithm CMNL. Based on these two modified algorithms, Algorithm DIMN can be modified as follows.

Modified Algorithm DIMN: Design of irredundant MOS networks for a given set of functions $f_1,\ldots,f_m$ with these output functions to be realized at the last level.

Step 1 Let $NFS^{o}(r+m;f_1,\ldots,f_m) = (u_1^*,\ldots,u_r^*;f_1,\ldots,f_m)$ and set i = 1 (If r is not known at this step, it will be determined after $NFS^{o}(r+m;f_1,\ldots,f_m)$ is obtained in Step 2 by applying the modified CMNL.)

Step 2 Using the modified Algorithm CMNL, obtain $NFS^{i-1}(r+m;f_1,\ldots,f_m) = (u_1,\ldots,u_{i-1},\underline{u}_i,\ldots,\underline{u}_r;f_1,\ldots,f_m)$.

Step 3 Using the modified Algorithm CMXL, obtain $\hat{NFS}^{i-1}(r+m;f_1,\ldots,f_m) = (u_1,\ldots,u_{i-1},\hat{u}_i,\ldots,\hat{u}_r;f_1,\ldots,f_m)$.

Step 4 Obtain function $\tilde{u}_i$ satisfying

$\tilde{u}_i(A) = 0$, if $\underline{u}_i(A) = \hat{u}_i(A) = 0$;

$\tilde{u}_i(A) = 1$, if $\underline{u}_i(A) = \hat{u}_i(A) = 1$; and

$\tilde{u}_i(A) = *$, if $\underline{u}_i(A) = 0$ and $\hat{u}_i(A) = 1$.

The function, $\tilde{u}_i$, obtained by this step is the MPF (Maximum Permissible Function) for the feasible $NFS^{i-1}(r+m;f_1,\ldots,f_m)$. After applying this step, the negative function sequence, $(u_1,\ldots,u_{i-1},\tilde{u}_i,u_{i+1}^*,\ldots,u_r^*;f_1,\ldots,f_m)$ is denoted as $\tilde{NFS}^{i-1}(r+m,f)$.

Step 5 Obtain an irredundant configuration for MOS cell g_i with function $\tilde{u}_i$ with respect to $x_1,\ldots,x_n,u_1,\ldots,u_{i-1}$. Let u_i be the function realized by this cell. Thus, $NFS^i(r+m,f) = (u_1,\ldots,u_{i-1},u_i, u^*_{i+1},\ldots,u^*_r; f_1,\ldots,f_m)$ has been obtained.

Step 6 If i = r, design an irredundant MOS cell configuration for each output gate g_{r+j} for function f_j with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_r$ for j = 1,...,m and terminate this algorithm; otherwise set i = i + 1 and go to Step 2.

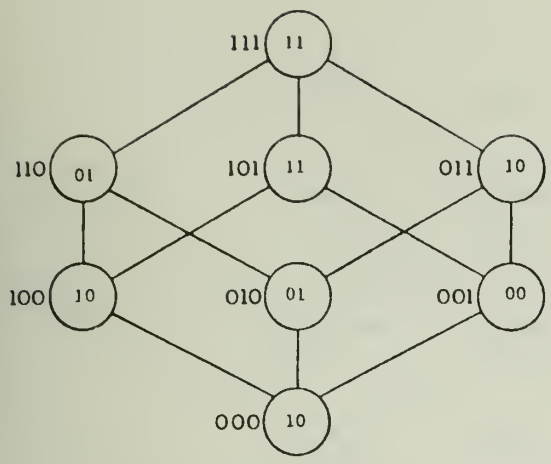
Similar to the Algorithm DIMN, this modified Algorithm DIMN applies at most r times of the loop consisting of Step 2 - Step 3 - Step 4 - Step 5 but the Step 5 of this algorithm alone must be executed m additional times in order to obtain the irredundant MOS cell configuration for $f_1,\ldots,f_m$.

Let us consider two examples with the same set of output functions. One shows the step-by-step design procedure for designing an irredundant MOS network with output functions realized at the last m levels, and the other shows the results with all output functions realized at the last level.

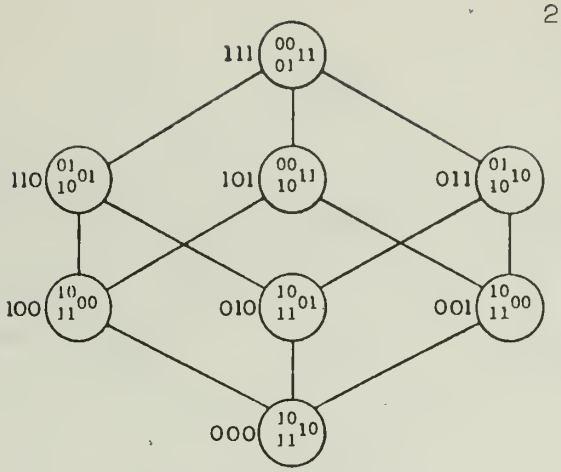
Example 2.2 Assume the m output functions $f_1,\ldots,f_m$ are realized at the last m levels. Let $P_1,\ldots,P_m$ be a set of numbers which determines the cell positions of output functions in such a way that $p_i = j$ if and only if the j-th cell (g_j) realizes the i-th function f_i. Then $NFS^o(R_f;f_1,\ldots,f_m; P_1,\ldots,P_m) = (u^*_1,\ldots,u^*_R)$ represents the partially specified negative function sequence for output functions with output cell positions $p_1,\ldots,P_m$. Two functions f_1 and f_2 are given in Fig. 2.5(a). Fig. 2.5(b) shows

$NFS^{\circ}(4;f_1,f_2;3,4) = (\underline{u}_1,\underline{u}_2,f_1f_2)$ and $\hat{NFS}^{\circ}(4;f_1,f_2;3,4) = (\hat{u}_1,\hat{u}_2,f_1,f_2)$. (upper and lower labels in vertices show $\underline{NFS}^{\circ}$ and $\hat{NFS}^{\circ}$, respectively.) By comparing $\underline{u}_1$ and $\hat{u}_1$, $NFS^1(4;f_1,f_2;3,4)_1$, $NFS^1(4;f_1,f_2;3,4)_2$, and $NFS^1(4;f_1,f_2;3,4)_3$ are obtained as shown in (c), (f), and (j), respectively. Next, by repeating the algorithm, $\underline{NFS}^1(4;f_1,f_2;3,4)_i$ and $\hat{NFS}^1(4;f_1f_2;3,4)_i$ are obtained for i = 1,2,3, as shown in (d), (g), and (k), respectively. $\tilde{NFS}^1(4;f_1,f_2;3,4)_i$ and $NFS^2(4;f_1,f_2;3,4)_i = NFS(4;f_1,f_2;3,4)_i$ are then derived, as shown in (e), (h), (i), (ℓ). The irredundant MOS networks corresponding to $NFS(4;f_1,f_2;3,4)_i$ are shown in (m), (n), and (o) for i = 1,2,3, respectively.

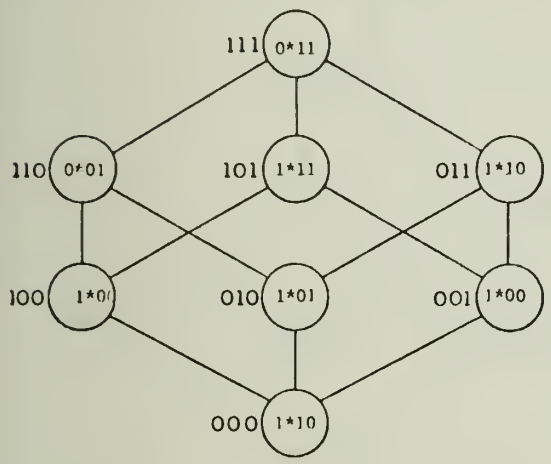
Example 2.3 All three solutions given in Example 2.2 have a connection from gate g_3 to g_4, i.e., the gate realizing function f_1 is not in the output level. Here, let us design an irredundant MOS network for f_1 and f_2 with the gates realizing these two functions at the last level. Fig. 2.6(a) shows $\underline{NFS}^{\circ}(4;f_1,f_2) = (\underline{u}_1,\underline{u}_2;f_1,f_2)$ and $\hat{NFS}^{\circ}(4;f_1f_2) = (\hat{u}_1\hat{u}_2;f_1f_2)$ obtained by Steps 2 and 3 of Modified Algorithm DIMN, respectively. Comparing $\underline{u}_1$ and $\hat{u}_1$ in Step 4, $\tilde{u}_1$ is obtained, and hence $\tilde{NFS}^{\circ}(4;f_1,f_2) = (\tilde{u}_1,u_2^*;f_1,f_2)$ is obtained as shown in (b). The only irredundant MOS cell configuration for $\tilde{u}_1$ is $u_1 = \overline{x_1x_3}$, being obtained by Step 5, and the corresponding $NFS^1(4;f_1f_2) = (u_1,u_2^*;f_1f_2)$ is shown in (c). Fig. 2.6(d) shows both $\underline{NFS}^1(4;f_1f_2) = (u_1,\underline{u}_2;f_1f_2)$ and $\hat{NFS}^1(4;f_1,f_2) = (u_1,\hat{u}_2;f_1,f_2)$ obtained by Steps 2 and 3, respectively. Since $\underline{u}_2(A) = \hat{u}_2(A)$ for all vertices A in the 3-cube; $\tilde{u}_2 = \underline{u}_2 = \hat{u}_2$ is completely specified, and the solution $NFS(4;f_1f_2)$ is obtained as shown in (e). The network corresponding to this NFS is obtained by designing irredundant MOS cell configurations for u_2, f_1, and f_2 and is shown in (f).



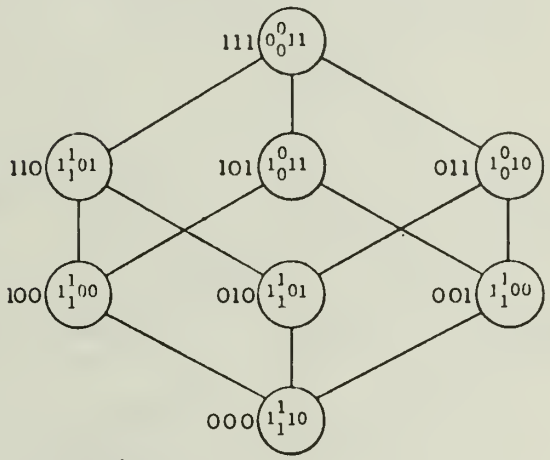
(a) $C_3(f_1,f_2)$.



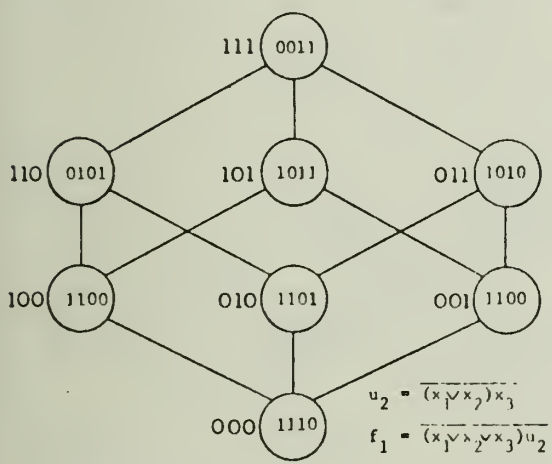
(b) $\underline{NFS}^0(4; f_1,f_2; 3,4) = (\underline{u}_1,\underline{u}_2, f_1,f_2)$.
$\widehat{NFS}^0(4, f_1,f_2; 3,4) = (\hat{u}_1,\hat{u}_2, f_1,f_2)$.



(c) $NFS^1(4; f_1,f_2; 3,4) = (\overline{x_1x_2}, u_2^*, f_1,f_2)$.



(d) $\underline{NFS}^1(4; f_1,f_2; 3,4)$.
$\widehat{NFS}^1(4; f_1,f_2; 3,4)$.

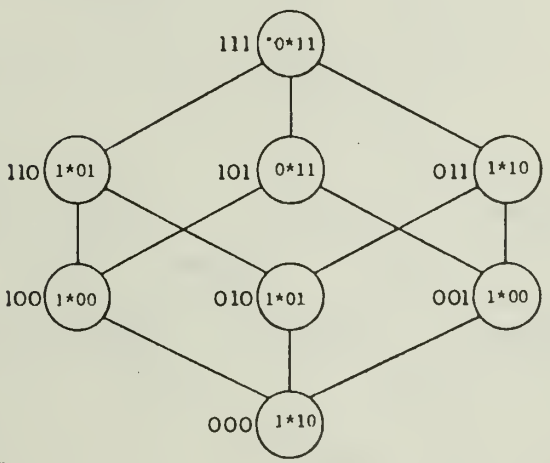


$u_2 = \overline{(x_1 \vee x_2)x_3}$

$f_1 = \overline{(x_1 \vee x_2 \vee x_3)u_2}$

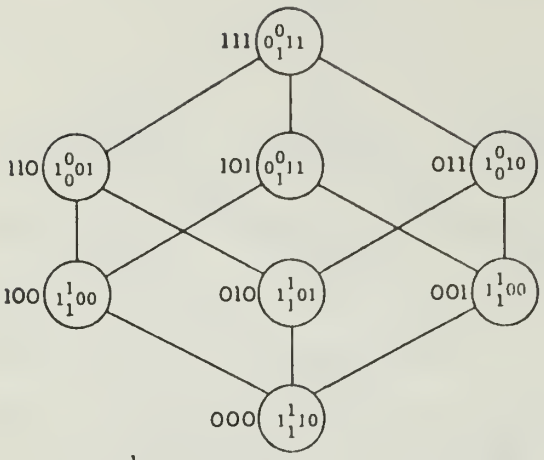
$f_2 = \overline{u_1(u_2(x_1 \vee x_3 \vee f_1) \vee x_2x_3)}$

(e) $\widehat{NFS}^1(4; f_1,f_2; 3,4) = NFS(4; f_1,f_2; 3,4) = (u_1,u_2, f_1,f_2)$.

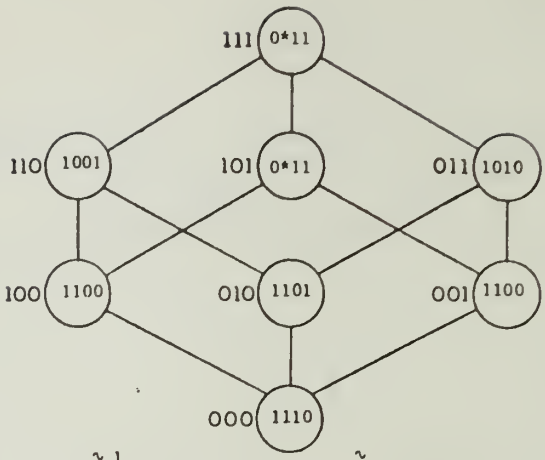


(f) $NFS^1(4; f_1,f_2; 3,4)_2 = (\overline{x_1x_3}, u_2^*, f_1,f_2)$.

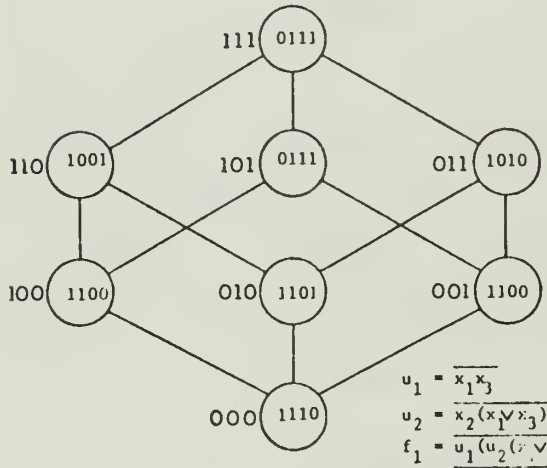
Fig. 2.5 Example 2.2.



(g) $\underline{NFS}^1(4;\ f_1,f_2;\ 3,4)_2 = (u_1,\underline{u}_2,f_1,f_2)$.
$\widehat{NFS}^1(4;\ f_1,f_2;\ 3,4)_2 = (u_1,\hat{u}_2,f_1,f_2)$.

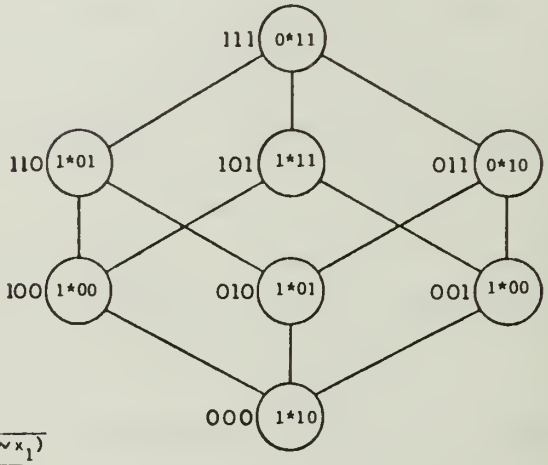


(h) $\tilde{NFS}^1(4;\ f_1,f_2;\ 3,4)_2 = (u_1\tilde{u}_2,\ f_1,f_2)$.

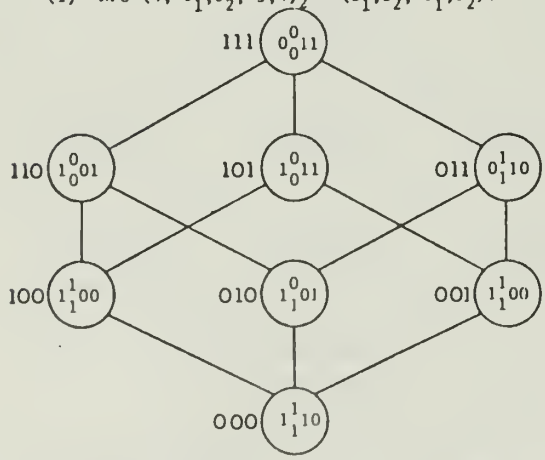


$u_1 = \overline{x_1 x_3}$
$u_2 = \overline{x_2(x_1 \vee x_3)}$
$f_1 = \overline{u_1(u_2(x_2 \vee x_3) \vee x_1)}$
$f_2 = \overline{u_1(f_1 \vee x_3 \vee x_1 u_2)}$

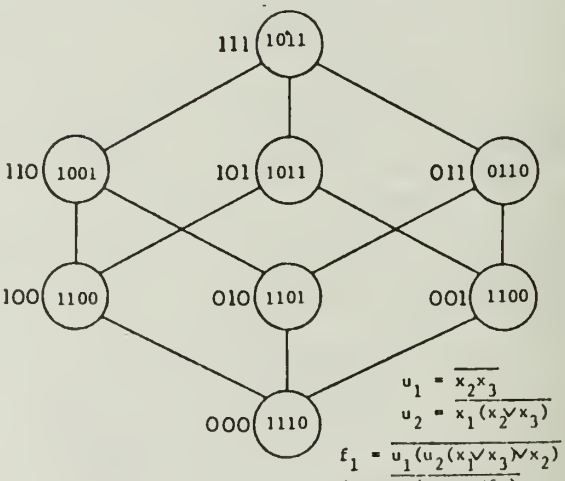
(i) $NFS^2(4;\ f_1,f_2;\ 3,4)_2 = (u_1,u_2,\ f_1,f_2)$.



(j) $NFS^1(4;\ f_1,f_2;\ 3,4)_3 = (\overline{x_2 x_3}, u_2^*, f_1, f_2)$.



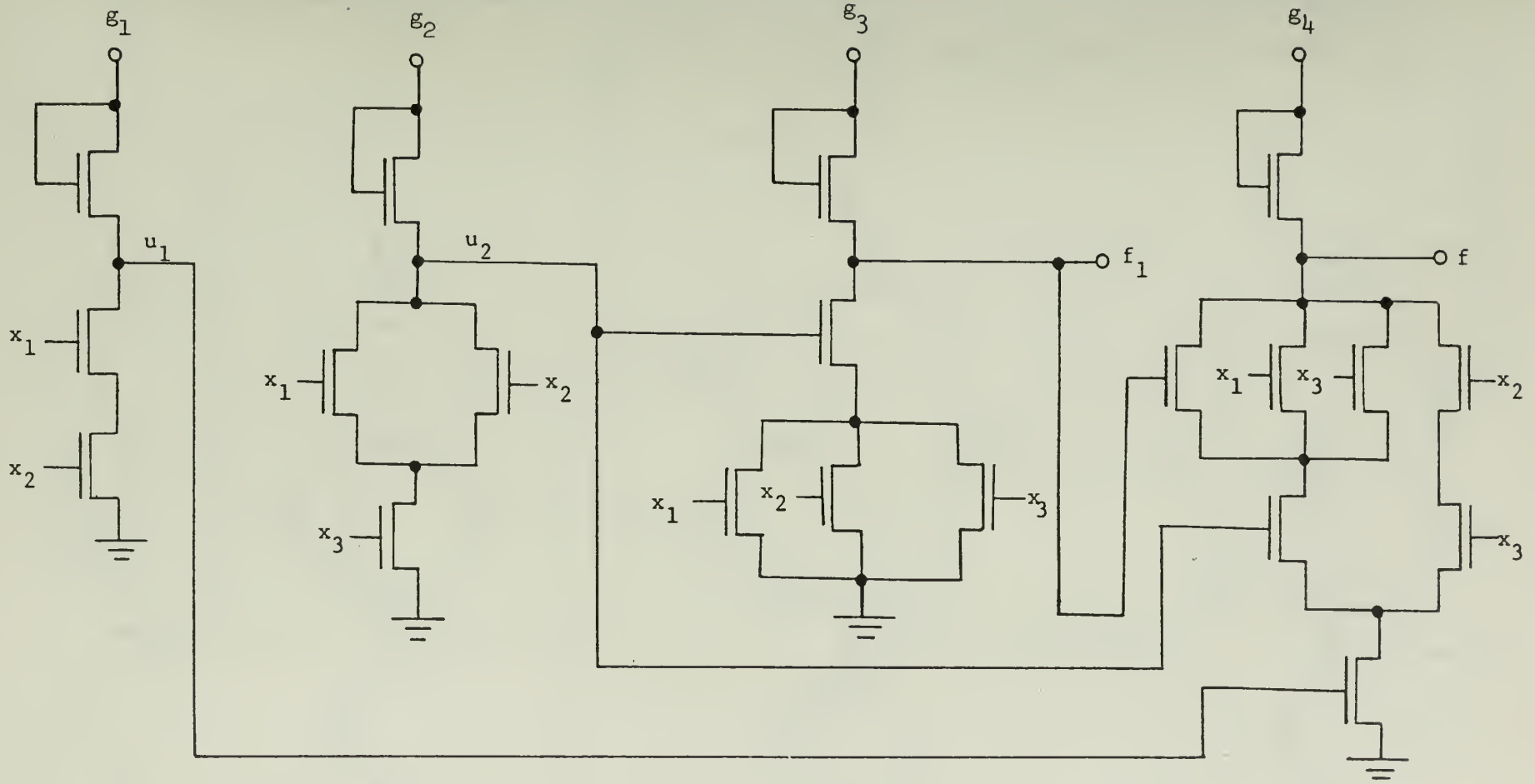
(k) $\underline{NFS}^1(4;\ f_1,f_2;\ 3,4)_3$.
$\widehat{NFS}^1(4;\ f_1,f_2;\ 3,4)_3$.



$u_1 = \overline{x_2 x_3}$
$u_2 = \overline{x_1(x_2 \vee x_3)}$
$f_1 = \overline{u_1(u_2(x_1 \vee x_3) \vee x_2)}$
$f_2 = \overline{u_2(x_1 \vee x_3 \vee f_1)}$

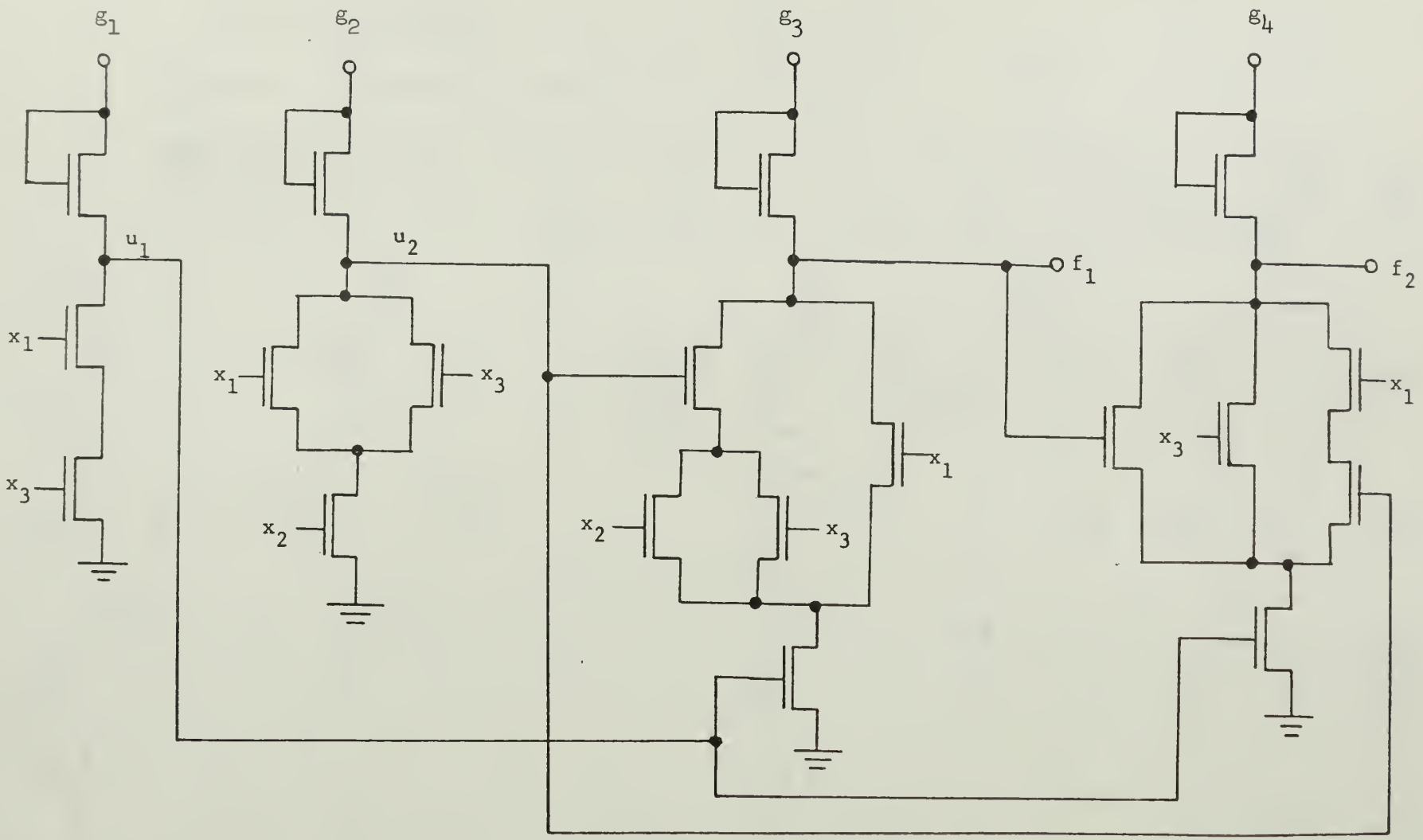
(l) $NFS^2(4;\ f_1,f_2;\ 3,4)_3 = NFS(4;\ f_1,f_2;\ 3,4)_3 = (u_1,u_2,\ f_1,f_2)$.

Fig. 2.5 (Continued)



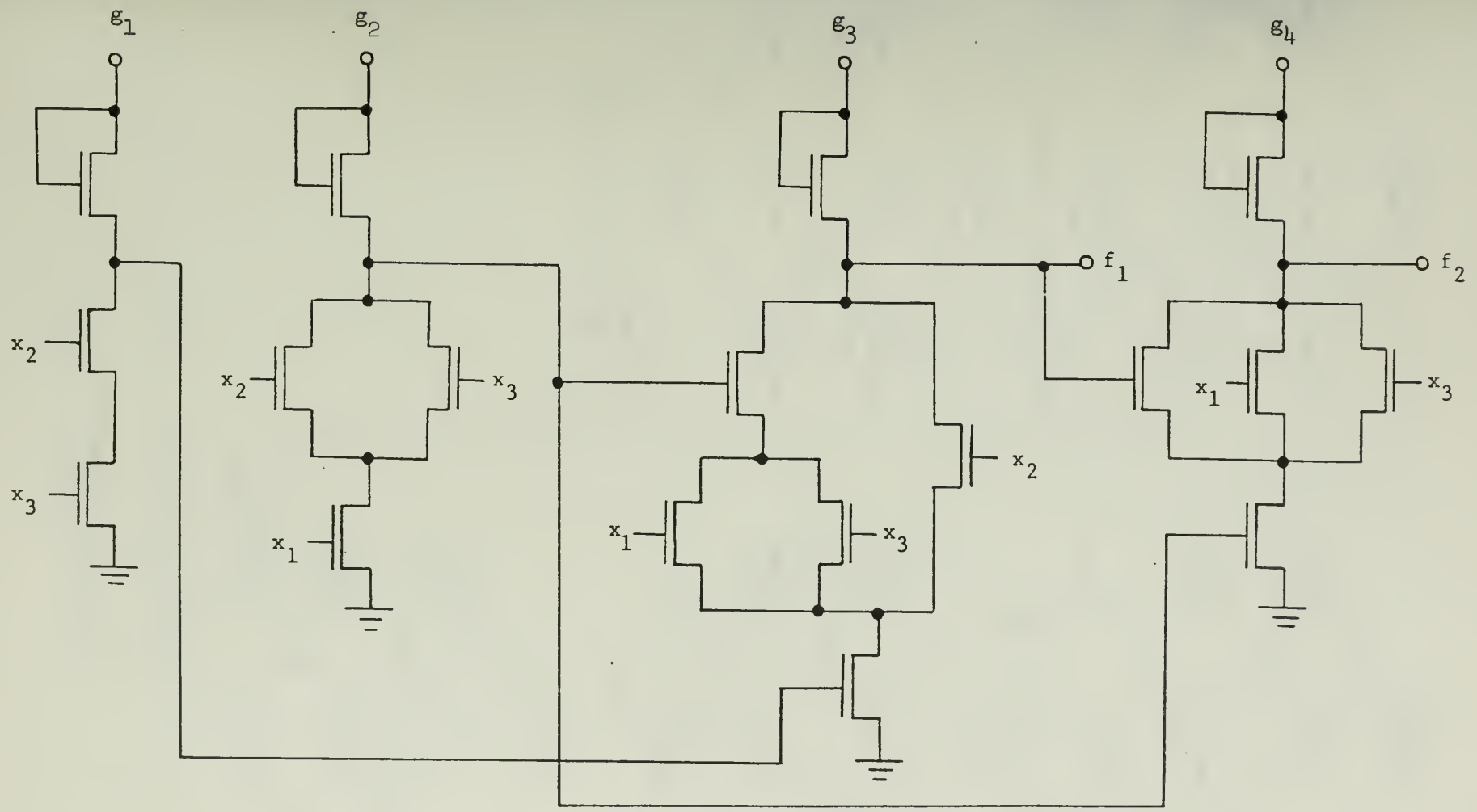
(m) Network corresponding to NFS(4; f_1,f_2; 3,4)$_1$ in (e).

Fig. 2.5 (Continued)



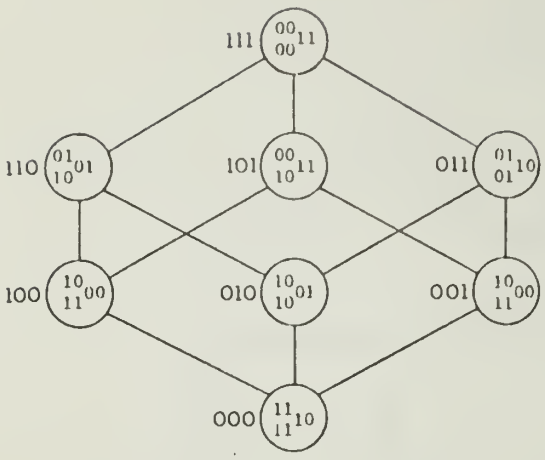
(n) Network corresponding to NFS(4; f_1,f_2; 3,4)$_2$ in (i).

Fig. 2.5 (Continued)

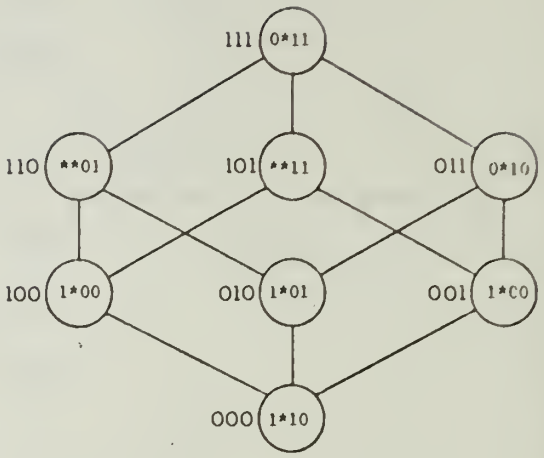


(o) Network corresponding to NFS(4; f_1, f_2; 3,4)$_3$ in (ℓ).

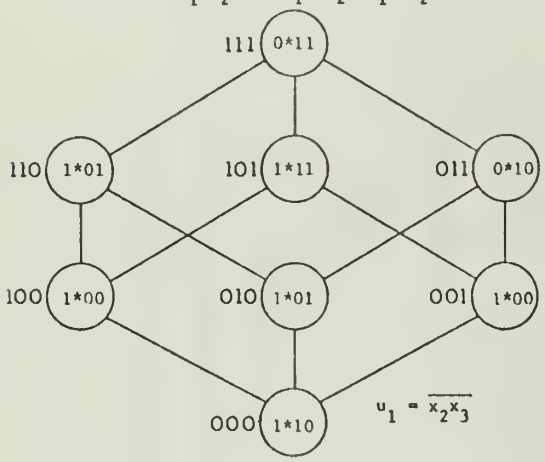
Fig. 2.5 (Continued)



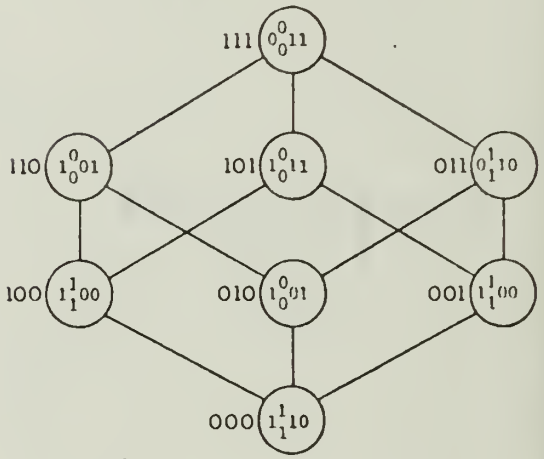
(a) $\underline{NFS}^0(2; f_1,f_2) = (\underline{u}_1, \underline{u}_2, f_1, f_2)$.
$\widehat{NFS}^0(2; f_1,f_2) = (\hat{u}_1, \hat{u}_2, f_1, f_2)$.



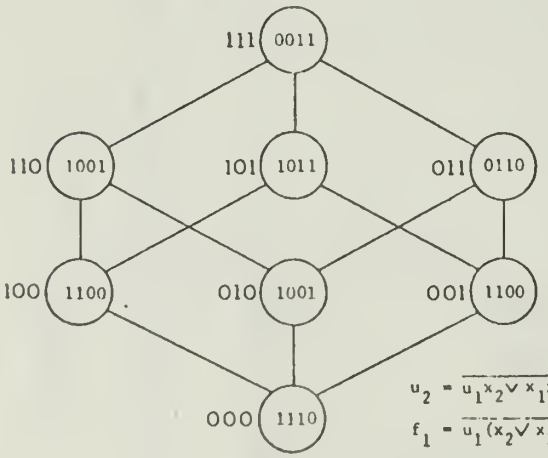
(b) $\tilde{NFS}^0(2; f_1,f_2) = (\tilde{u}_1, u_2^*; f_1,f_2)$.



(c) $NFS^1(2; f_1,f_2) = (u_1, u_2^*; f_1, f_2)$.

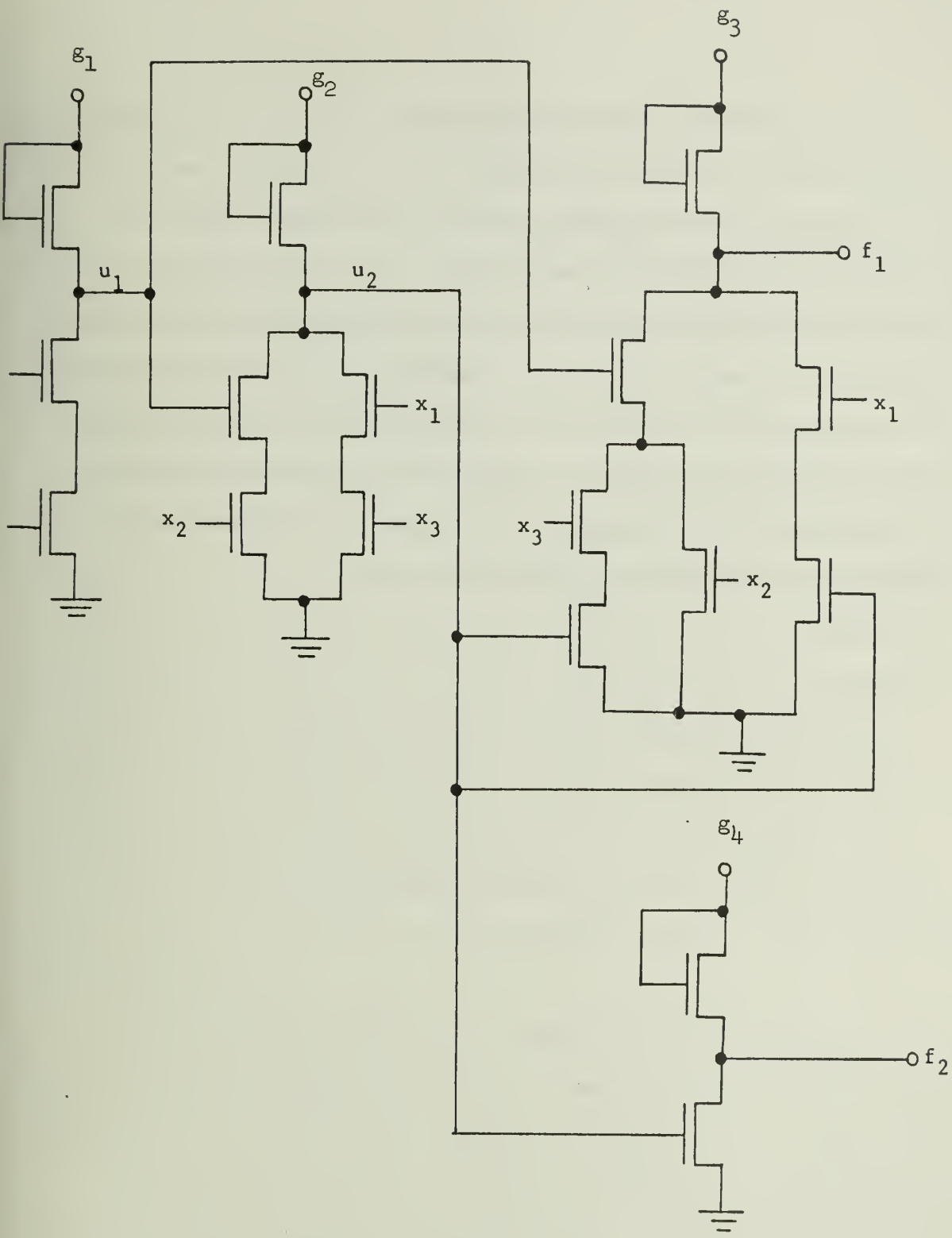


(d) $\underline{NFS}^1(2; f_1,f_2) = (u_1,\underline{u}_2; f_1,f_2)$.
$\widehat{NFS}^1(2; f_1,f_2) = (u_1,\hat{u}_2; f_1,f_2)$.



(e) $\widehat{NFS}^1(2; f_1,f_2) = NFS^2(2; f_1,f_2) = NFS(2; f_1,f_2)$.

Fig. 2.6 Example 2.3



(f) Network corresponding to NFS(2; f_1,f_2) of (e).

Fig. 2.6 (Continued)

It is interesting to see that the network in Fig. 2.6(f) consists of only 17 FETs (including load MOSFETs) while the networks shown in Fig. 2.5 (m), (n) and (o) consist of 19, 20 and 18 FETs, respectively. The networks designed with all output gates at the last level tend to contain more FETs than the networks designed with all output gates at the last m levels. However, the results obtained by the later case depend on the output cell positions of the output functions $f_1,\ldots,f_M$. With different output cell positions, different MOS network configurations are usually produced. Usually, we do not know which permutation for the later case will result in better MOS networks.

3. PROCEDURE DIMN

This chapter will discuss FORTRAN program DIMN for designing irredundant MOS networks. The program DIMN consists of two main parts: one part is for implementing the Algorithm DIMN for finding irredundant MOS networks in the cases of single-output function and multiple-output functions with all the output functions realized at the last m levels and the other part is for implementing the Modified Algorithm DIMN for finding irredundant MOS networks in the multiple-output functions case with all the output functions realized at the last level. The flowchart of Algorithm DIMN is shown in Fig. 3.0.1 and the flowchart of Modified Algorithm DIMN is shown in Fig. 3.0.2. A flag FL is used to distinguish these two cases for designing irredundant MOS networks with multiple-output functions. The flowchart in Fig. 3.0.1 or Fig. 3.0.2 will be applied according to whether FL is set to 0 or 1, respectively. For the case of incompletely-specified functions (single-output or multiple-output), Fig. 3.0.1 and Fig. 3.0.2 are slightly modified. The modifications will not be shown.

The program DIMN is written in FORTRAN for IBM 360 (also Cyber 175). The entire program requires 100 K bytes of core storage, about 64 K being occupied by the actual program instructions and 36 K by the stored data.

In the following, Section 3.1 describes internal data representation, that is, the representation of a labeled N-cube. Although the internal data representation is exactly the same as that described in Yamamoto's thesis, the description will be given in Section 3.1 in order to explain

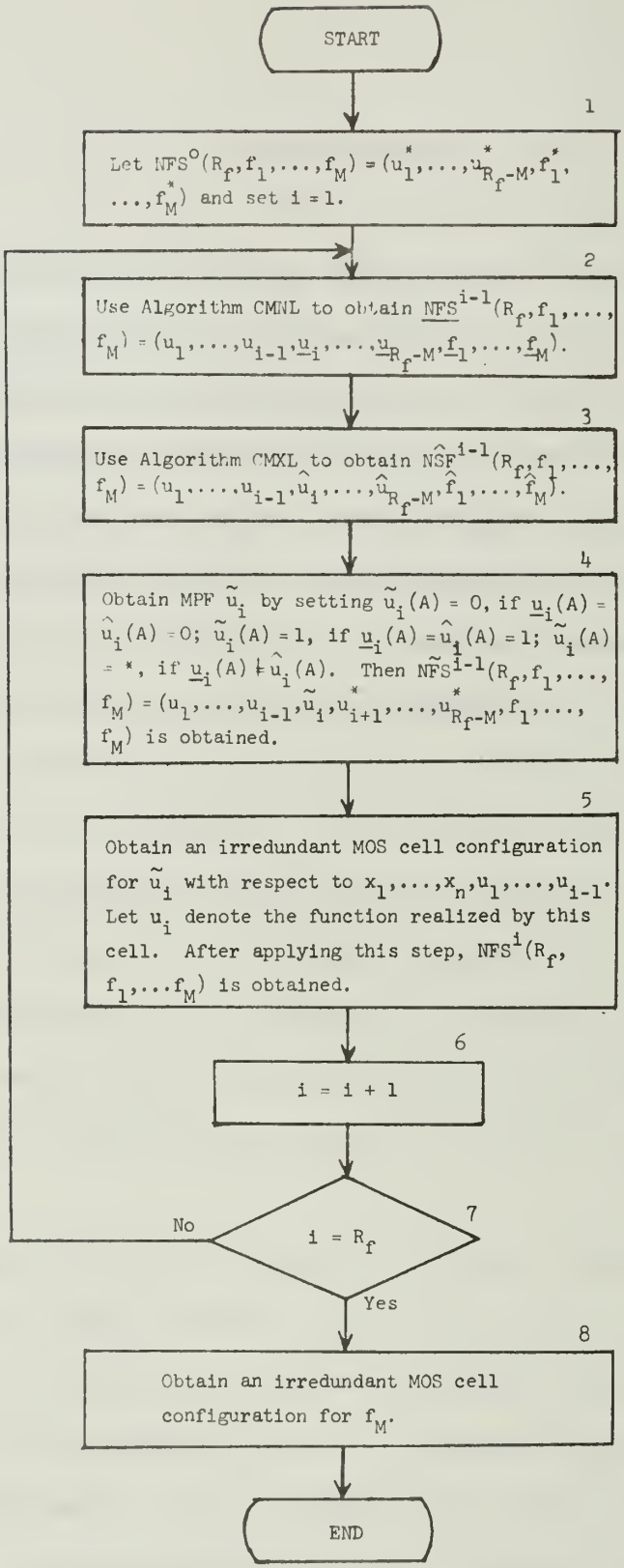


Fig. 3.0.1 Flowchart of Algorithm DIMN.

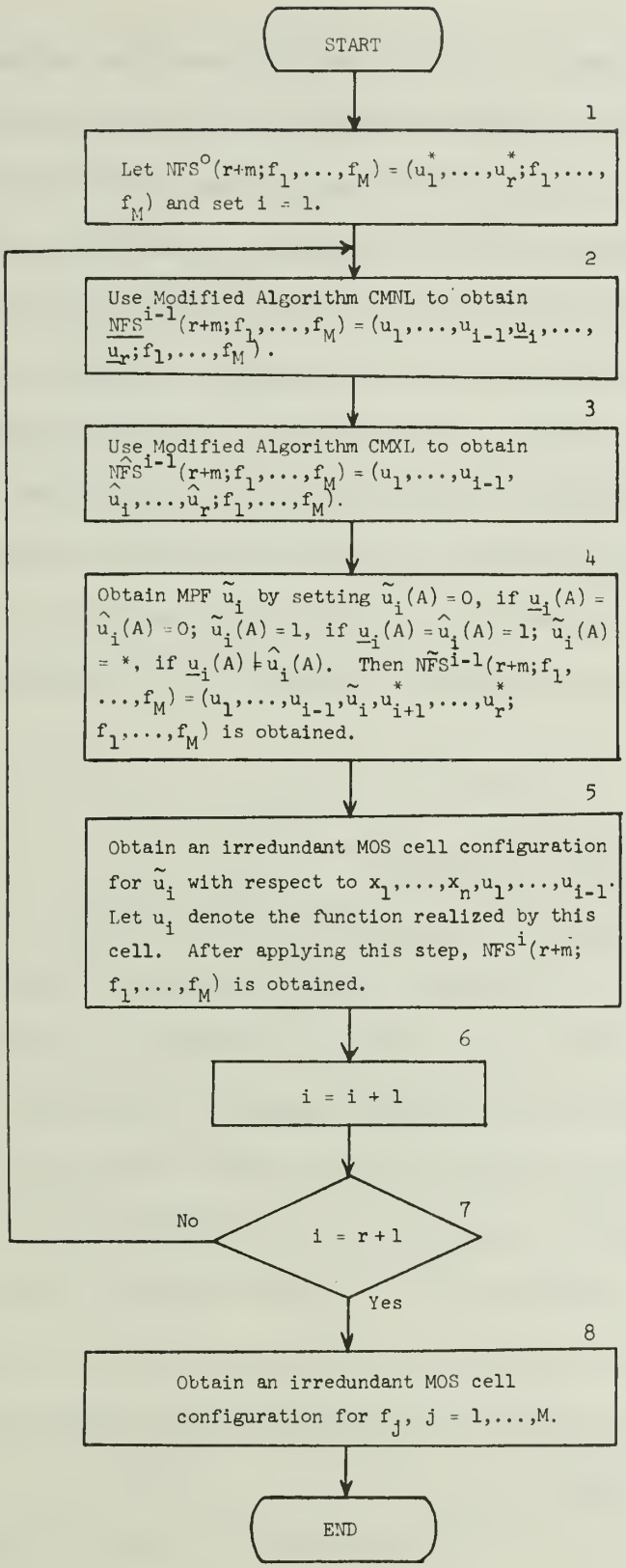


Fig. 3.0.2 Flowchart of Modified Algorithm DIMN.

the flowchart in Section 3.3. Section 3.2 describes the general organization of program DIMN and Section 3.3 explains the modifications of subprocedures in detail. All other subprocedures which were explained in detail in Yamamoto's thesis [5] will not be repeated in this paper.

The following notations and explanations will be used throughout the program DIMN and this paper, N is the number of external input variables, M is the number of output functions and R_F is the number of MOS cells obtained by this program, where R_F is R_f used in the previous chapters, F here denoting the previous f, II is a pointer which indicates the iteration of the program loop for determining one MOS cell configuration. For example, if II is three, the third MOS cell configuration is going to be determined.

3.1. Internal Data Representation

In order to implement DIMN program (The flowcharts of this program are shown in Fig. 3.0.1 and Fig. 3.0.2), two kinds of labeled N-cubes are required. One is the labeled N-cube for storing input data, and internal results, and obtaining MPF (Maximum Permissible Function). The other is the labeled N-cube for obtaining an irredundant MOS cell configuration. The labeled N-cube for obtaining MPF is accessed in blocks 2, 3, and 4 in both Fig. 3.0.1 and Fig. 3.0.2. In the comments of the program listing in Appendix B, this labeled N-cube is simply referred to as the N-cube and each vertex is assigned a vertex number which represents the input vector in binary form.

As shown in Fig. 3.1.1, each vertex in a N-cube has the following

five fields; LABEL, DCARE, MNL, MXL and CHAIN. LABEL field, together with DCARE field, stores the partially specified negative function sequence (The notation, $NFS^{i-1}(R_F,f_1,\ldots,f_M)$, shown in Fig. 3.0.1 or the notation, $NFS^{i-1}(r+m;f_1,\ldots,f_M)$, shown in Fig. 3.0.2). MNL and MXL fields store $\underline{NFS}^{i-1}(R_F,f_1,\ldots,f_M)$(or $\underline{NFS}^{i-1}(r+m;f_1,\ldots,f_M)$) that is the

VERTEX NUMBER

LABEL	DCARE	MNL	MXL	CHAIN

Fig. 3.1.1 The vertex in the N-cube.

completion of $NFS^{i-1}(R_F,f_1,\ldots,f_M)$(or $NFS^{i-1}(r+m;f_1,\ldots,f_M)$) by CMNL, and $\hat{NFS}^{i-1}(R_F,f_1,\ldots,f_M)$(or $\hat{NFS}^{i-1}(r+m;f_1,\ldots,f_M)$) that is the completion of $NFS^{i-1}(R_F,f_1,\ldots,f_M)$(or $NFS^{i-1}(r+m;f_1,\ldots,f_M)$) by CMXL, respectively. CHAIN field stores the link to the next vertex in the list of the vertices of the same weight. In general, the vertex with vertex number (j-1) contains the j-th elements of these arrays as shown in Fig. 3.1.2. This is because the index of an array in FORTRAN cannot be zero. Therefore, vertex 0, that is, the vertex with binary value 0 corresponds to the first elements of the arrays LABEL, DCARE, MNL, MXL and

VERTEX (j-1)

LABEL(j)	DCARE(j)	MNL(j)	MXL(j)	CHAIN(j)

Fig. 3.1.2 The vertex with vertex number (j-1).

CHAIN; vertex 1, that is, the vertex with binary value 1 corresponds to the second elements of these arrays and so on. An example of the 3-cube is shown in Fig. 3.1.3. Fig. 3.1.4 is the actual representation of this 3-cube in the computer memory. As can be seen in Fig. 3.1.3, the first and the last vertices in the lists of the vertices with the same weight are pointed by pointer arrays STARTL and ENDL, respectively. For example, vertex 3, the first vertex in the list of the vertices with weight 2 is pointed by pointer STARTL(3) and vertex 6, the last vertex in the same list is pointed by pointer ENDL(3). In general, the first and the last vertices in the list of the vertices with weight K is pointed by pointers STARTL(K+1) and ENDL(K+1), respectively.

INDEX	LABEL	DCARE	MNL	MXL	CHAIN
(1)					
(2)					3
(3)					5
(4)					6
(5)					
(6)					7
(7)					
(8)					

	STARTL	ENDL
(1)	1	1
(2)	2	5
(3)	4	7
(4)	8	8

Fig. 3.1.4 Computer internal representation of the 3-cube in Fig. 3.1.3.

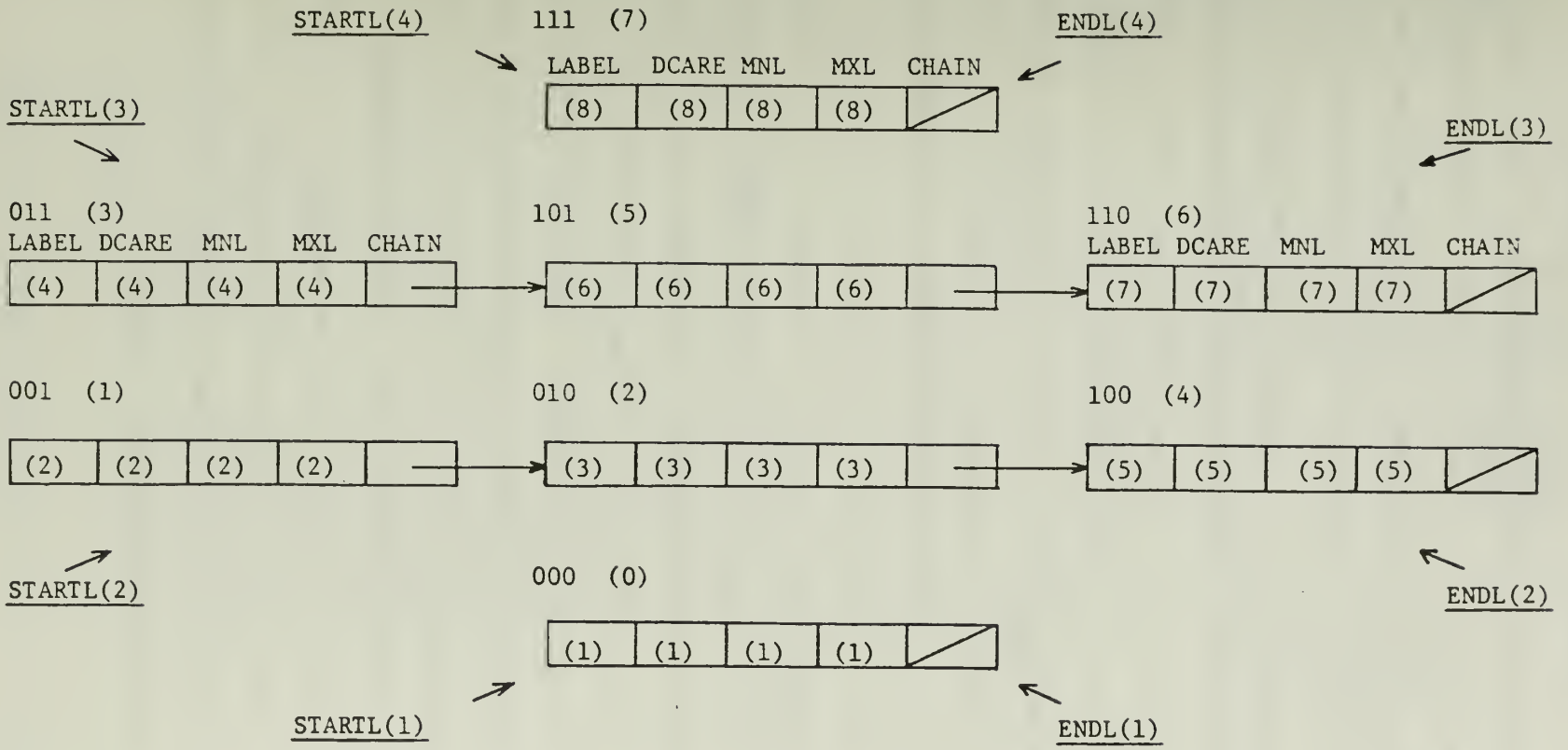


Fig. 3.1.3 Data structure of the 3-cube.

The labeled N-cube for obtaining an irredundant MOS cell configuration is accessed in block 5 in Fig. 3.0.1 and Fig. 3.0.2. In the comments of the program listing in Appendix B, this labeled N-cube is referred to as the Large-cube because the dimension N of this N-cube, starting from the initial value n, is increased by one, each time the program loop in Fig. 3.0.1 or Fig. 3.0.2 is executed. However, the increases of the dimension of this Large-cube are different for the case in Fig. 3.0.1 and the case in Fig. 3.0.2. In Fig. 3.0.1 (i.e., the cases of single-output function and multiple-output functions with all output functions realized at last M levels), the dimension of the Large-cube, starting from n, is increased by one each time this program loop is executed until the final value (i.e., the dimension of the Large-cube at the last iteration of this program loop), $N + R_F - 1$, is reached. On the other hand, the case of multiple-output functions with all output functions realized at the last level, the dimension of the Large-cube, starting from n, is increased by one each time the program loop in Fig. 3.0.2 is executed until the value $N + r$ is reached. After this iteration, the dimension of the Large-cube will not be increased and remains the same value (N+r) for realizing the output functions f_i for $i = 1,\ldots,M$ (This will be discussed in detail in Section 4.2). The above discussion implies that a new Large-cube with different contents has to be construced each time block 5 in Fig. 3.0.1 or in Fig. 3.0.2 is executed. Unlike the Large-cube, the previously mentioned N-cube, once being constructed at the begining of the program, will never be changed.

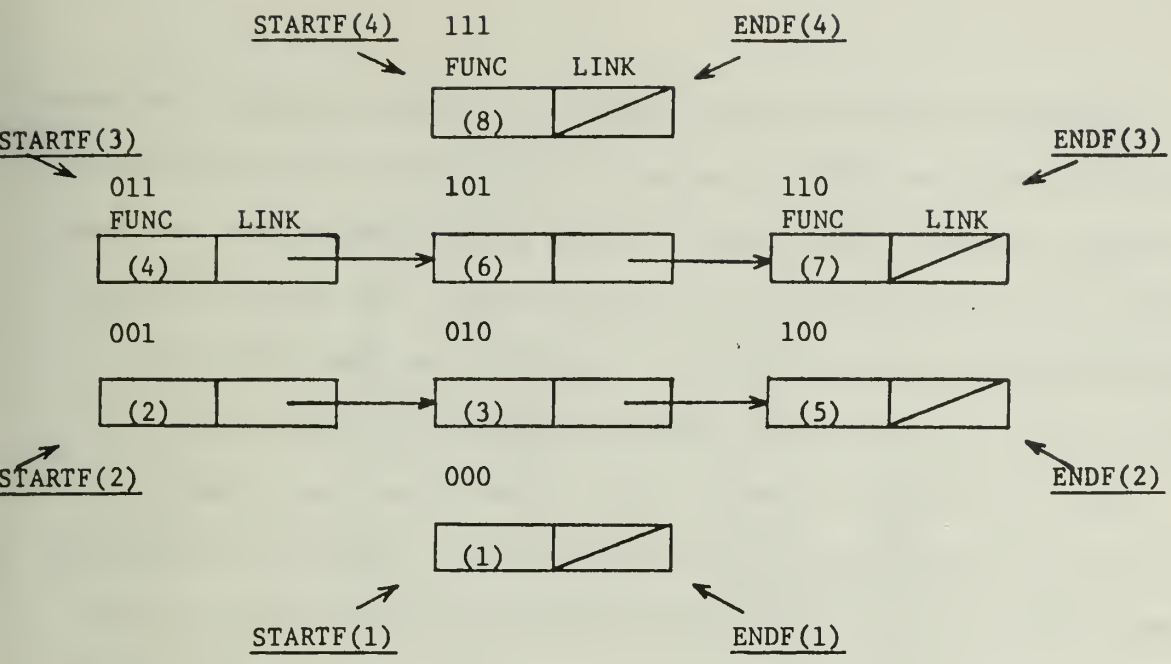


Fig. 3.1.5 3-dimensional Large-cube.

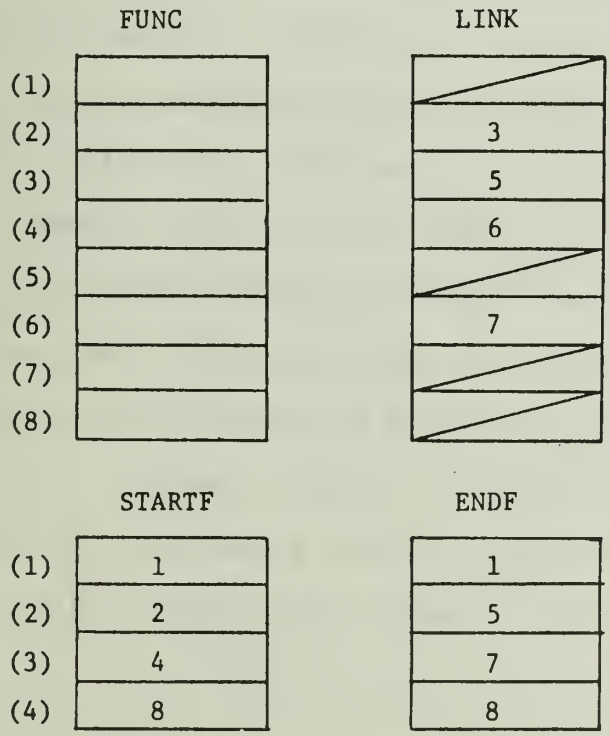


Fig. 3.1.6 Internal representation of 3-dimensional Large-cube in Fig. 3.1.5.

Similar to the N-cube, each vertex in the Large-cube which consists of two fields, FUNC and LINK, is assigned a vertex number which represents the input vector in binary form. The LINK field corresponds to the CHAIN field of the N-cube and points to the next vertex with the same weight. MPF (Maximum Permissible Function) is stored in the FUNC field after MPF is obtained at the end of block 4 in Fig. 3.0.1 or Fig. 3.0.2.

Fig. 3.1.5 shows an example of the 3-dimensional Large-cube and Fig. 3.1.6 is the internal representation of the 3-dimensional Large-cube shown in Fig. 3.1.5. As can be seen in Fig. 3.1.5, two pointer arrays, STARTF and ENDF, correspond to those two pointer arrays, STARTL and ENDL, in Fig. 3.1.3.

3.2 General Organization of Program DIMN

This section shows the general organization of program DIMN and outlines each subprogram. Fig. 3.2.1 describes the general organization program DIMN. This program consists of subroutine MAIN and the following subroutines: INPUT, NCUBE, CMNL, CMXL, MPF, IMC, ASFUN1, ASFUN2, INCMNT, DSCAN, ASIGN1, DCRMNT, CMPR and PRINT. In order to include the Modified Algorithm DIMN in this program, three subroutines, MAIN, CMNL, and CMXL have to be modified and will be explained in greater detail in next section. The other subroutines were discussed in detail in Yamamoto's thesis.
In Fig. 3.2.1, an arrow from block i to block j means that the subroutine represented by block i calls the subroutine represented by block j.

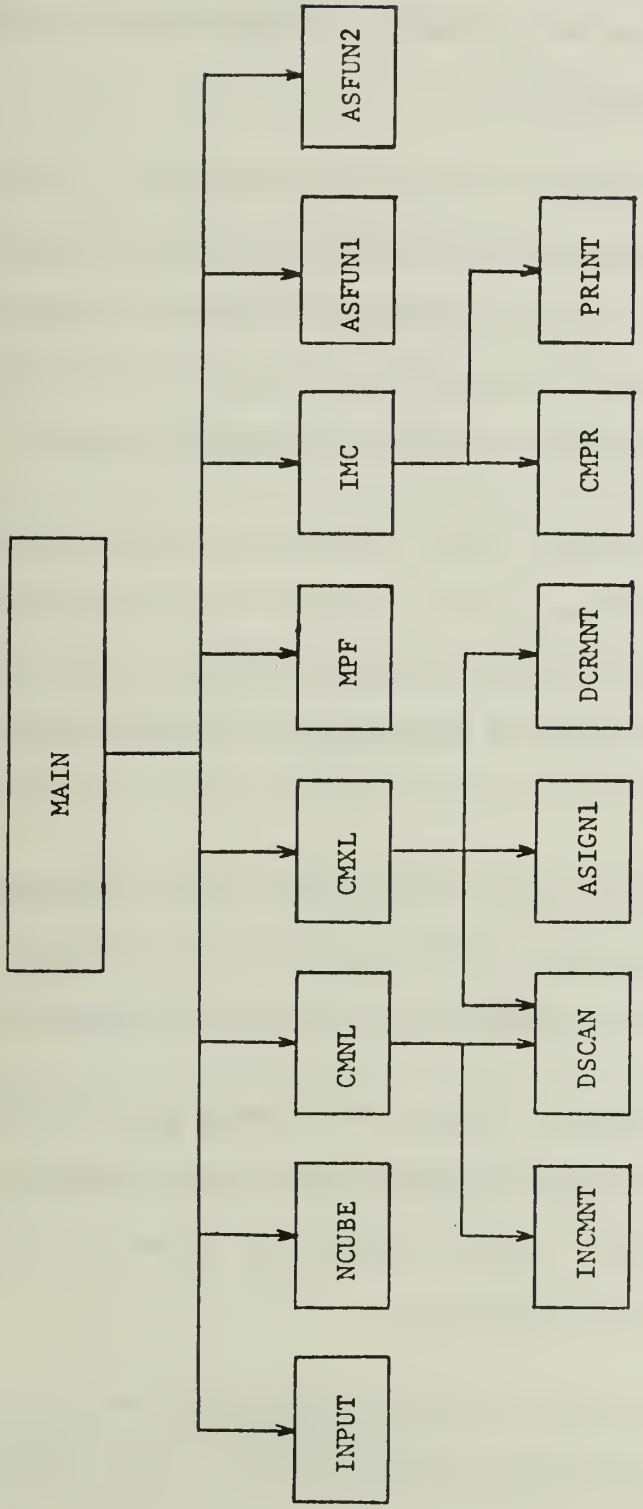


Fig. 3.2.1 General organization of program DIMN.

Subroutine INPUT: This subroutine reads in an N-cube input data, i.e., network parameters and given output functions. Input data setup is described in detail in Chapter 4.

Subroutine NCUBE: This subroutine constructs the N-cube for the input data. If the number of external input variables is N, this N-cube will contain 2^N vertices. Subroutine NCUBE will examine the input vector in binary form which is to be assigned to each vertex in the N-cube one by one and will link the vertices with the same weight together.

Subroutine CMNL: This subroutine calls subroutines INCMNT and DSCAN to execute the conditional minimum labeling based on the output function values read into the N-cube. R_F, the number of MOS cells which is required to realize the given function(s) is determined at the first execution of this subroutine CMNL.

Subroutine CMXL: This subroutine executes the conditional maximum labeling based on the output function values read into the N-cube. Subroutines DSCAN, ASIGN1 and DCRMNT are called by this subroutine.

Subroutine MPF: This subroutine obtains the maximum permissible function from the results obtained in MNL and MXL fields in the N-cube by subroutines CMNL and CMXL, respectively. Then this subroutine stores the resulting MPF in the FUNC field of the Large-cube.

Subroutine IMC: This subroutine obtains an irredundant MOS cell configuration from the maximum permissible function in the Large-cube

obtained by subroutine MPF. This subroutine calls subroutines CMPR and PRINT whenever necessary.

Subroutine ASFUN1: Under certain circumstances (e.g., while realizing the last gate), the maximum permissible function does not need to be obtained. This means that the program loop which consists of CMNL, CMXL and MPF need not be executed and by detecting such circumstances, the computation time can be saved. This subroutine is called under such circumstances. The function values stored in LABEL field in the N-cube are directly stored by this subroutine in the FUNC field in the Large-cube.

Subroutine ASFUN2: This subroutine is called when MNL is identical to MXL at some iteration i of the program loop. The values in the MNL field in the N-cube are stored by this subroutine in FUNC field in the Large-cube.

Subroutine DSCAN: This subroutine determines the number of don't care bits in the function part of the label assigned to each vertex in the N-cube and the weight assigned to these don't care bits.

Subroutine MAIN: This subroutine calls the subroutines shown in Fig. 3.2.1, whenever necessary, and implements Algorithm DIMN and Modified Algorithm DIMN.

3.3 Descriptions about Subroutines

In this section, three modified subroutines, MAIN, CMNL and CMXL, are explained. Although the explanation of the variables and arrays appearing

in these subroutines can be found in the program listing in Appendix B, some variables (which are not defined in [5]) are defined in this section in order to explain each flowchart.

(1) Subroutine MAIN (Fig. 3.3.1)

In block 1, parameters N, M and FL are read in and if these parameters do not satisfy the restriction on problem size (discussed in detail in Section 4.2), an error message is printed and the program execution is terminated. This block also tests EOF (end of file) condition and if all data have been exhausted, the program execution is terminated.

In block 2, the LABEL and DCARE fields in the N-cube are initialized to zero.

In block 3, the subroutine INPUT is called and values on output function cards are read into the LABEL and DCARE fields in the N-cube. A logical variable, CS, is set to "true" if all values on output function cards are completely specified; otherwise, CS is set to "false".

In block 4, the N-cube is constructed based on the external variables, N.

In block 5, II, the pointer which indicates the number of iterations of the program loop is initialized to 1. One MOS cell configuration is determined each time the loop is executed.

Block 6 executes conditional minimum labeling (CMNL) and at the first execution of this block the value of R_F (the minimum number of MOS cells) is determined.

In block 7, the obtained R_F value is examined and if it is equal to the number of given output functions, that is, all the given output functions are negative functions, the maximum permissible function (MPF) need not

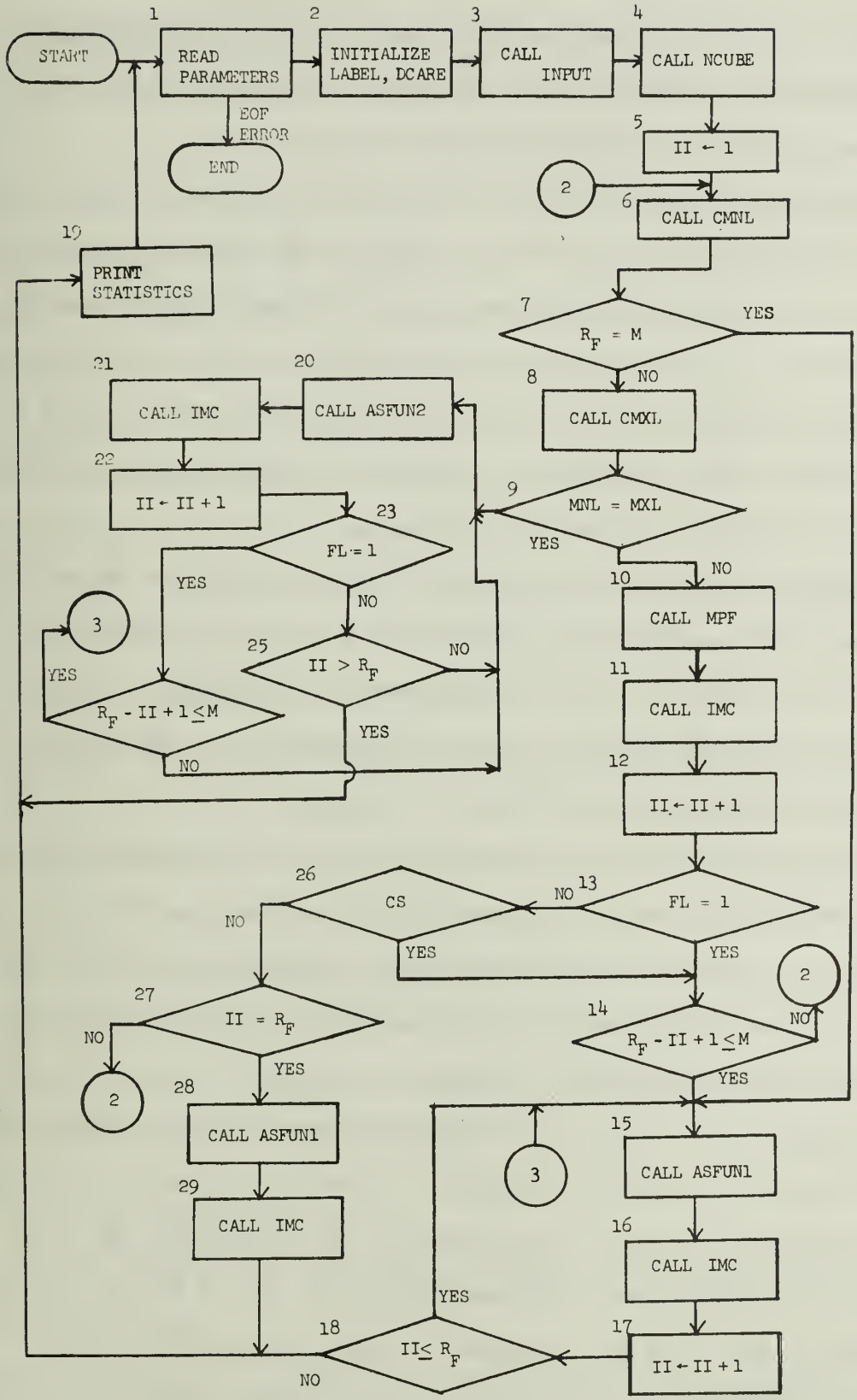


Fig. 3.3.1 Flowchart of Subroutine MAIN.

be obtained; the MOS cell configuration can be obtained immediately by calling subroutines ASFUN1 and IMC. If $R_F \neq M$, the subroutine CMXL is called to implement conditional maximum labeling.

In block 9, the labels assigned to the N-cube by CMNL and CMXL are compared and if the labels assigned by CMNL and CMXL take the same value at every vertex in the N-cube, there exists the unique minimum negative gate network (i.e., unique maximum permissible function) and block 9 is immediately followed by the loop (Block 20, 21, ...) for obtaining an irredundant MOS cell configuration. Under this condition, the execution of subroutines CMNL, CMXL and MPF is skipped.

In block 10 and block 11, subroutine MPF and subroutine IMC are called if the labels assigned by CMNL and CMXL to the N-cube take different values at some vertices. After the implementation of block 11, the configuration for the MOS cell indicated by pointer II is determined (i.e., the II-th MOS cell is obtained).

In block 12, Pointer II is increased by one. FL is examined in block 13 and if FL is equal to 1 (multiple-output functions with all output functions to be realized at the last level), block 14 is executed; otherwise (multiple-output functions with all output functions to be realized at the last M levels), block 26 is executed.

In block 14, if we are not going to determine the output gates, the program control is returned to block 6. If the M output gates are going to be determined, the loop of the procedures for determining MOS cell configuration is immediately followed (Blocks 15, 16, 17 and 18) and the procedure for obtaining MPF is skipped. After applying this loop, an irredundant MOS network which realizes the given output functions is obtained.

In block 27, if II is not equal to R_F, in other words, if we are not going to determine the configuration of the last MOS cell in the network with all output functions realized at the last M levels, the program control is returned to block 6. If II is equal to R_F, that is, if the configuration of the last MOS cell is going to be determined, blocks 28 and 29 are followed to determine this configuration of the last MOS cell.

In block 19, the statistics about the obtained network are printed out and the program control is transferred to block 1. The contents of the statistics are described in detail in Chapter 5.

In block 1, if problem cards are not exhausted, a new problem is read in and the entire process is repeated. If problem cards have already been exhausted, the program is terminated.

In block 26, the logical variable CS is examined and if CS is true, this block is followed by block 14. In the case of multiple-output functions, the maximum permissible function need not be obtained for the output function which is completely specified. Therefore, if one of the MOS cells which realize the completely specified output functions is being considered, block 14 is immediately followed by the loop for determining MOS cell configuration and the process for obtaining the maximum permissible function is skipped.

(2) Subroutine CMNL (Fig. 3.3.2)

The following is the definition of variables and arrays which appear in this subroutine.

USPFY: As mentioned in Section 3.1, array LABEL stores a partially specified negative function sequence. USPFY stores the number of unspecified

bits in array LABEL. The relation between USPFY, R_F, II and M is shown in Fig. 3.3.3.

W: This variable stores an actual weight plus one.

PTRB: This is a pointer to vertex B to which the minimum possible value is to be assigned.

PTRA: This is a pointer to vertex A which is connected to vertex B by the edge from vertex A to vertex B. The vertex A has a greater weight

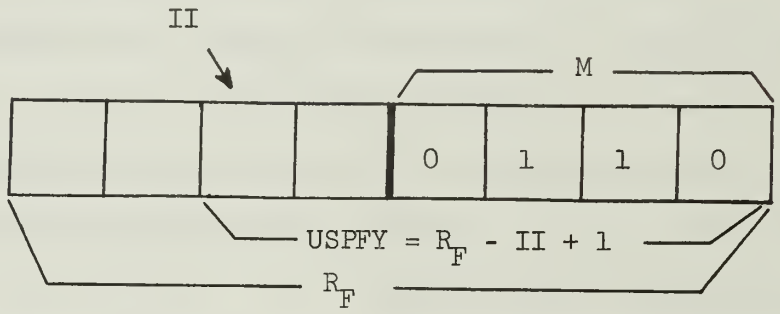


Fig. 3.3.3 Relation between USPFY, R_F, II and M for an element of array LABEL.

than vertex B by one.

LBX: This is a variable which stores LABEL field of vertex pointed by pointer B (For the case where FL = 0).

LAX: This is a variable which stores the minimum label assigned to vertex A (For the case where FL = 0).

LBY: This is a variable which stores DCARE field of vertex pointed by pointer B (to be examined in subroutine DSCAN). The number of don't care bits and the weight assigned to each don't care bit are determined.

NDCARE: This is a variable which stores the number of don't care bits. This value is determined in subroutine DSCAN.

BWEIT: This is an array which stores the weight assigned to each don't care bit.

PTRBX: This is a variable which stores the input vector in binary form assigned to vertex B (to be examined bit by bit). PRTA will be determined from this variable and variable SHIFT.

INB: This is a counter for the increment of LBX (used for the case where FL = 0).

LBZ: This is a variable which stores the first part of label (i.e., $(u_1,\ldots,u_r)$) for vertex B (used for the case where FL = 1).

LAZ: This is a variable which stores the first part of label for vertex A (used for the case where FL = 1).

L: This is a variable which stores the weight of output function bits in array LABEL.

J: This is a variable which stores the index value of array BWEIT.

LBT: This is a variable which stores the first part of label together with the first (M-L) bits of output functions stored in LABEL field for vertex. B. This variable is used for examining the (M-L)th function value at vertex B under the condition that FL = 1.

LAT: This is a variable which has the same meaning as that of LBT except being used to describe vertex A.

LB: This is a variable which stores the first part of a label together with one of the output function bits. This function bit is determined by L such that LB represents the sequence $(u_1,\ldots,u_r,f_i)$ for i = M - L.

LA: This is a variable which has the same meaning as that of LB except being used to describe vertex A.

INC: This is a counter for the increment of LB (used for that case we where FL = 1).

Fig. 3.3.4 shows the structures for LBZ, and LBT and Fig. 3.3.5 shows the structure of LB, for some vertex B in the N-cube.

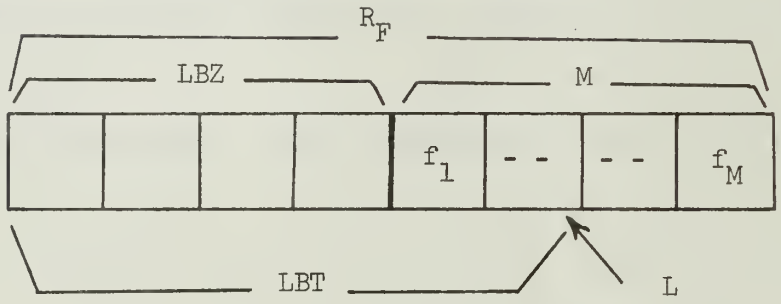


Fig. 3.3.4 The structures of LBZ and LBT for the LABEL field of vertex B.

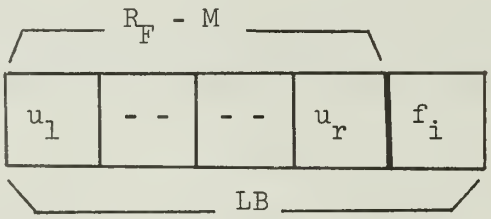


Fig. 3.3.5 The internal representation of LB for vertex B, where i = 1,...,M. Each time LB is constructed, i is equal to M -L.

The following is the detailed explanation of Fig. 3.3.2(a) and (b). In block 1 (Fig. 3.3.2(a)), the minimum possible label is assigned to the vertex with weight N. Since LABEL is initialized to zero in block 2 in subroutine MAIN, the unspecified bits in LABEL(STARTL(N+1)) have already been assigned zero, the minimum possible value. As shown in Fig. 3.3.3, the number of unspecified bits is obtained by calculating R_F - II + 1. Although R_F is obtained after the first implementation of subroutine CMNL, the value of R_F has to be specified in subroutine MAIN before CMNL is called for the first time.

Let us consider FL = 0 first. After assigning a label to the vertex with weight N, a minimum possible label is asigned to every other vertex in the following way. In the loop consisting of blocks 4 through 16, each vertex pointed by PTRB (vertex B) is assigned a minimum label. In general, as shown in Fig. 3.3.6, there exist several vertices which are directly connected to vertex B by edges. Minimum labels which have already been

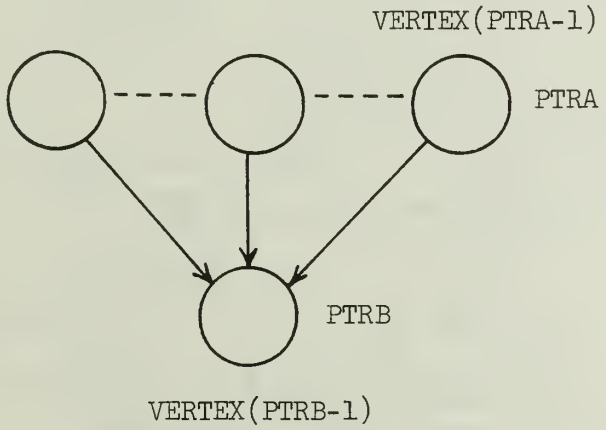


Fig. 3.3.6 Relation between vertices A and B in CMNL.

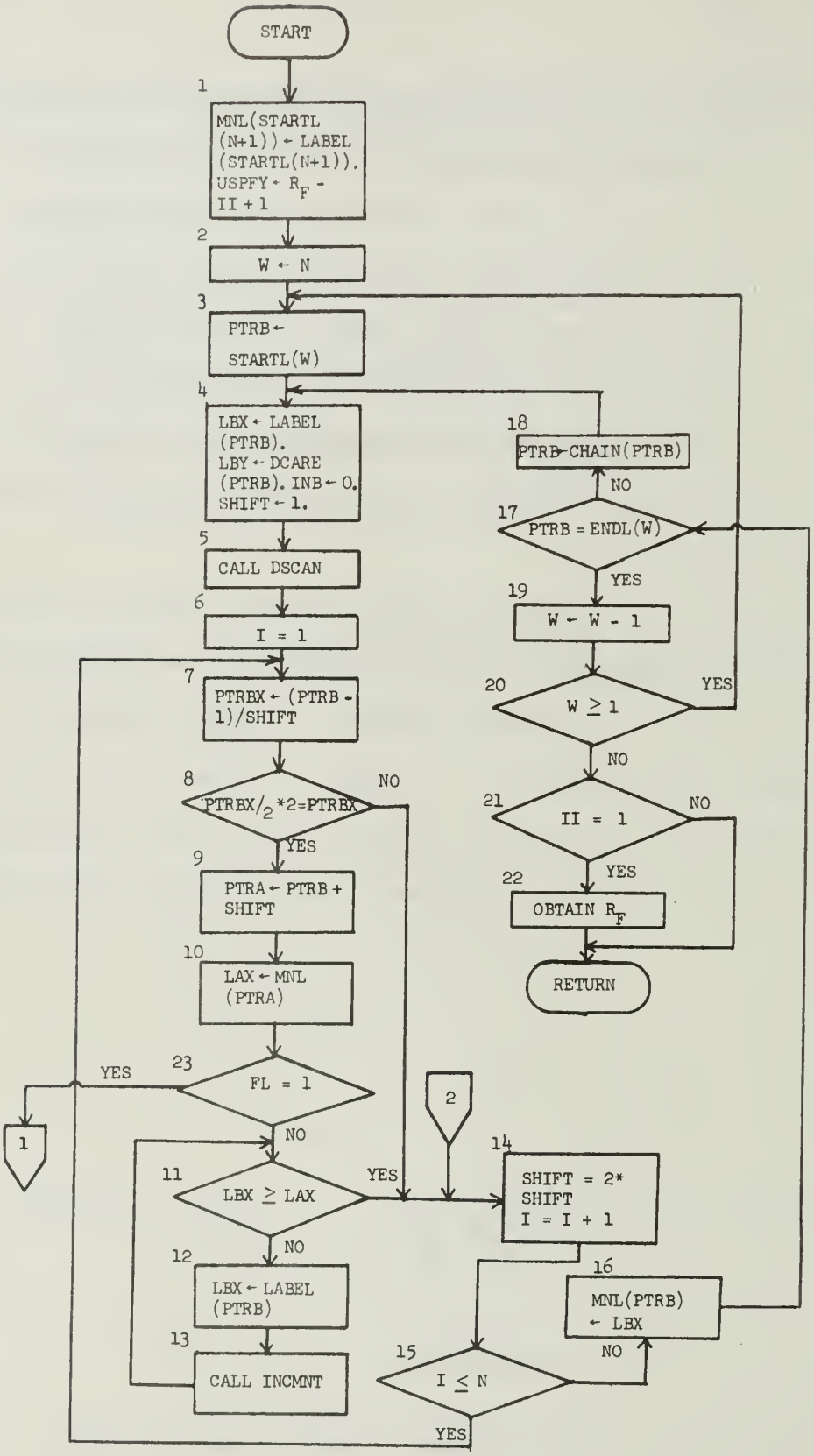


Fig. 3.3.2(a) Flowchart of subroutine CMNL.

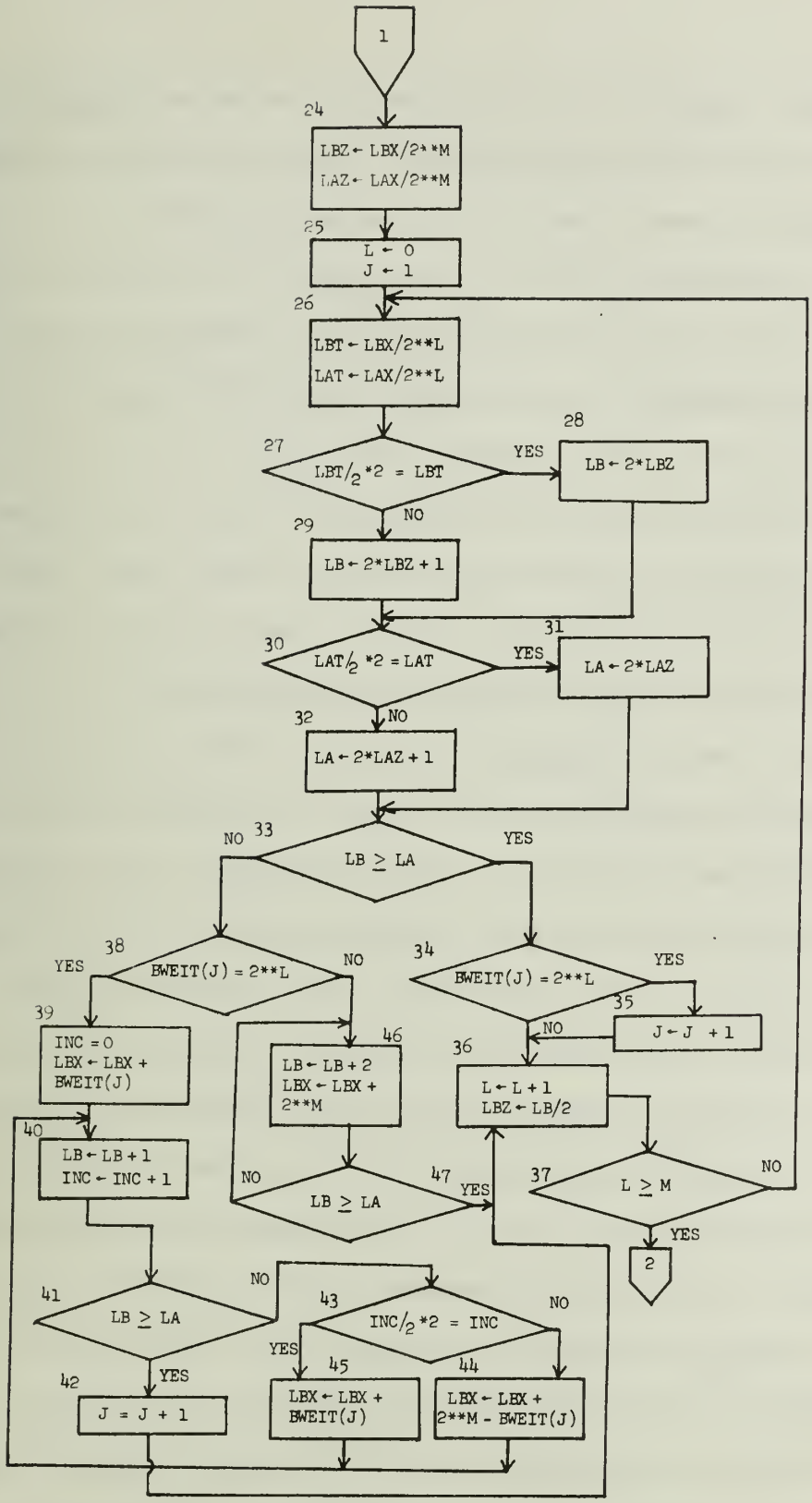


Fig. 3.3.2(b) Flowchart of subroutine CMNL (continued).

assigned to these vertices are compared with LBX one by one (block 11) and LBX is increased by one (block 13) until none of the labels assigned to these vertices is larger than the label assigned to vertex B. In block 9, PTRA is calculated by PTRA = PTRB + SHIFT. As vertices with the same weight, say (w-1), are linked by CHAIN field, starting from the vertex pointed by STARTL(W), all the vertices within the linked list can be exhausted by this scheme (block 3, 17 and 18).

Starting from the vertices with weight (N-1), the above procedure is applied to every set of vertices with the same weight until a label is assigned to the vertex with weight zero. The value of R_F is obtained (blocks 21 and 22) at the first execution of this subroutine.

When FL is 1, the loop (blocks 11, 12 and 13) in Fig. 3.3.2(a) is replaced by the whole flowchart in Fig. 3.3.2(b). The main difference between these two cases is the way to compare the minimum label which has been assigned to every vertex A with the label assigned to the corresponding vertex B. In Fig. 3.3.2(b), the output function values in LABEL field for each vertex are considered as a vector. This means that LB (i.e., $(u_1(B),\ldots,u_r(B),f_i)$ of vertex B) has to be greater than or equal to every LA (i.e., $(u_1,(A),\ldots,u_r(A),f_i)$ for every vertex A connected to this vertex B), for $i = 1,\ldots,M$. One of the vertices which directly connect to vertex B and have weight greater by one than that of vertex B is obtained in block 9. Therefore, the main task in Fig. 3.3.2(b) is to compare M LB's of vertex B with the corresponding M LA's of this vertex A one by one and LB is increased by one repeatedly until none of the LA's assigned to this vertex A is larger than the corresponding LB assigned to vertex B. Then, the program control is transferred to block 14

in Fig. 3.3.2(a) and a next vertex A is obtained (block 7) and the same process in Fig. 3.3.2(b) is repeated until all possible vertex A is exhausted. The program control is then transferred to block 16.

In Fig. 3.3.2(b), blocks 24,...,32 obtain one LB of vertex B and the corresponding LA of vertex A. Block 33 compares these two values. If LB is greater than or equal to LA, the loop including block 34 is executed; otherwise, block 38 is executed. If the last bit in LB is originally a don't care of output function values (tested in block 34), the corresponding weight of this don't care bit is skipped in block 35 (since LB is greater than or equal to LA, this don't care bit needs not be set to one). In block 36, the weight of output function values is increased by one and the first part of label (i.e., $(u_1,\ldots,u_r)$) assigned to this LB is stored back to LBZ. If all the possible LA's of this vertex A are exhausted, the program control is transferred to block 14 in Fig. 3.3.2(a); otherwise the program control is returned to block 26.

As mentioned above, if LA is not smaller than LB, block 38 is executed. If the last bit in LB is not originally a don't care of output function values, LB is increased by two (i.e., increase the first part of label by one) each time the loop consisting of blocks 46 and 47 is executed until LB is greater than or equal to LA. Then, the program control is transferred to block 36 and the next pair of LB and LA is examined. If the last bit of LB is originally a don't care, this don't care bit is set to one (in block 39) and the counter, INC, is increased by one (in block 40). Then, block 41 compares LB and LA. If LA is not smaller than LB, LB is increased by one each time the loop consisting of blocks 43, 44 or 45, 40 is

executed until LA becomes smaller than LB, and then the index of array BWEIT is increased and the program control is transferred to block 36.

(3) Subroutine CMXL (Fig. 3.3.7)

This subroutine is executed in almost the same way as the subroutine CMNL is. The flowchart in Fig. 3.3.7(b) (i.e., for the case of FL = 1) is the same as that in Fig. 3.3.2(b), except that after 1's are assigned to the unspecified bits of the label field of vertex B by calling subroutine ASIGN1, the label in subroutine CMXL is decreased instead of being increased in subroutine CMNL.

The following is the definitions of variables used in this subroutine. Variables with the same definition as those in subroutine CMNL are omitted.

PTRB: This is a pointer to vertex B to which the maximum possible value is to be assigned.

PTRA: This is a pointer to vertex A which is connected to vertex B by edge. The vertex B has a larger weight than vertex A by one. The relation between vertex A and vertex B is shown in Fig. 3.3.8.

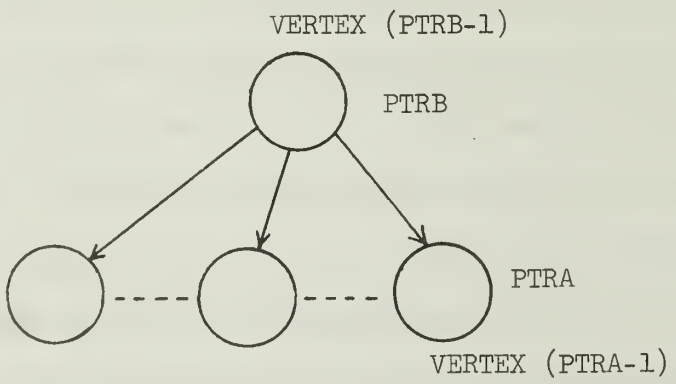


Fig. 3.3.8 Relation between vertices A and B in CMXL.

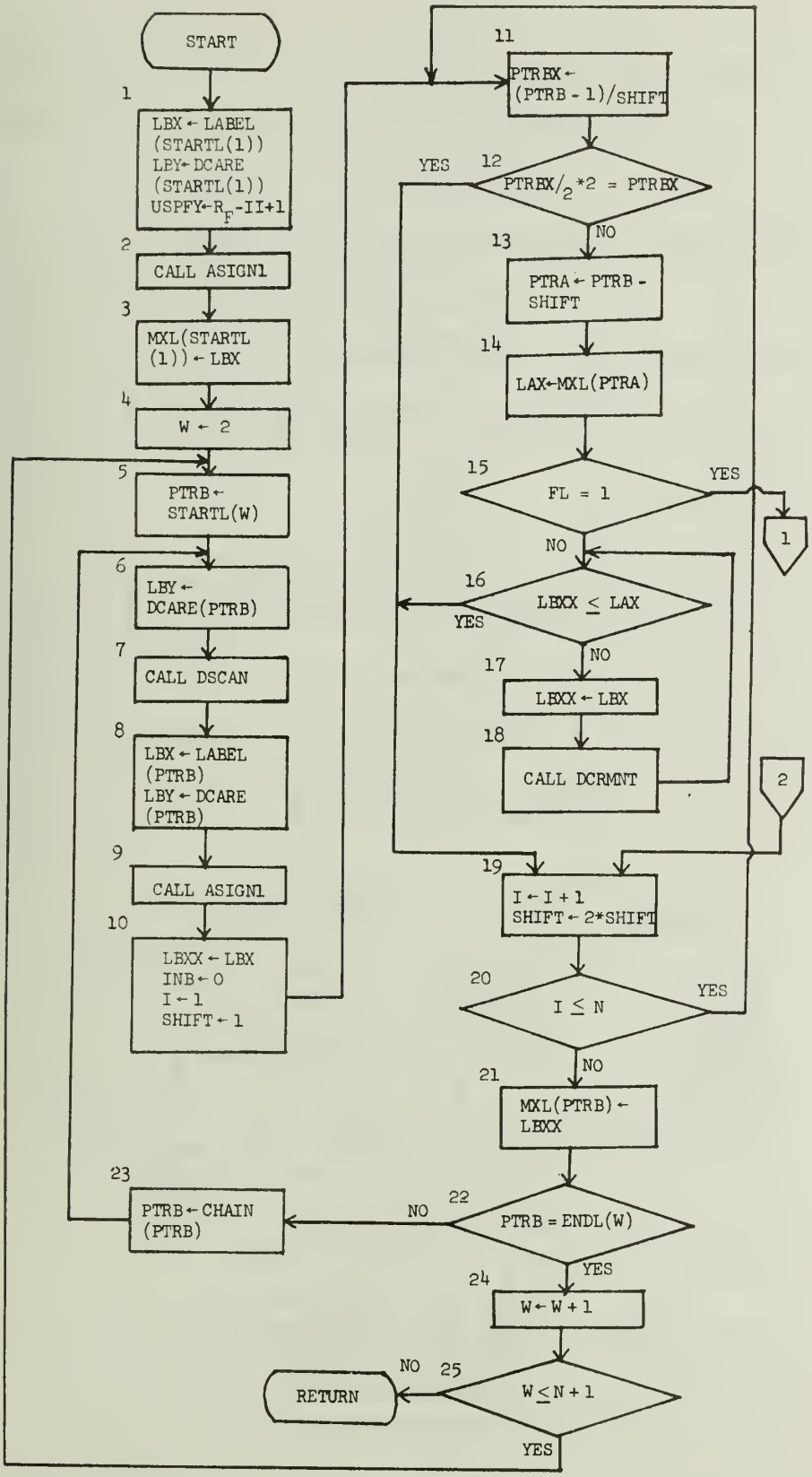


Fig. 3.3.7(a) Flowchart of subroutine CMXL.

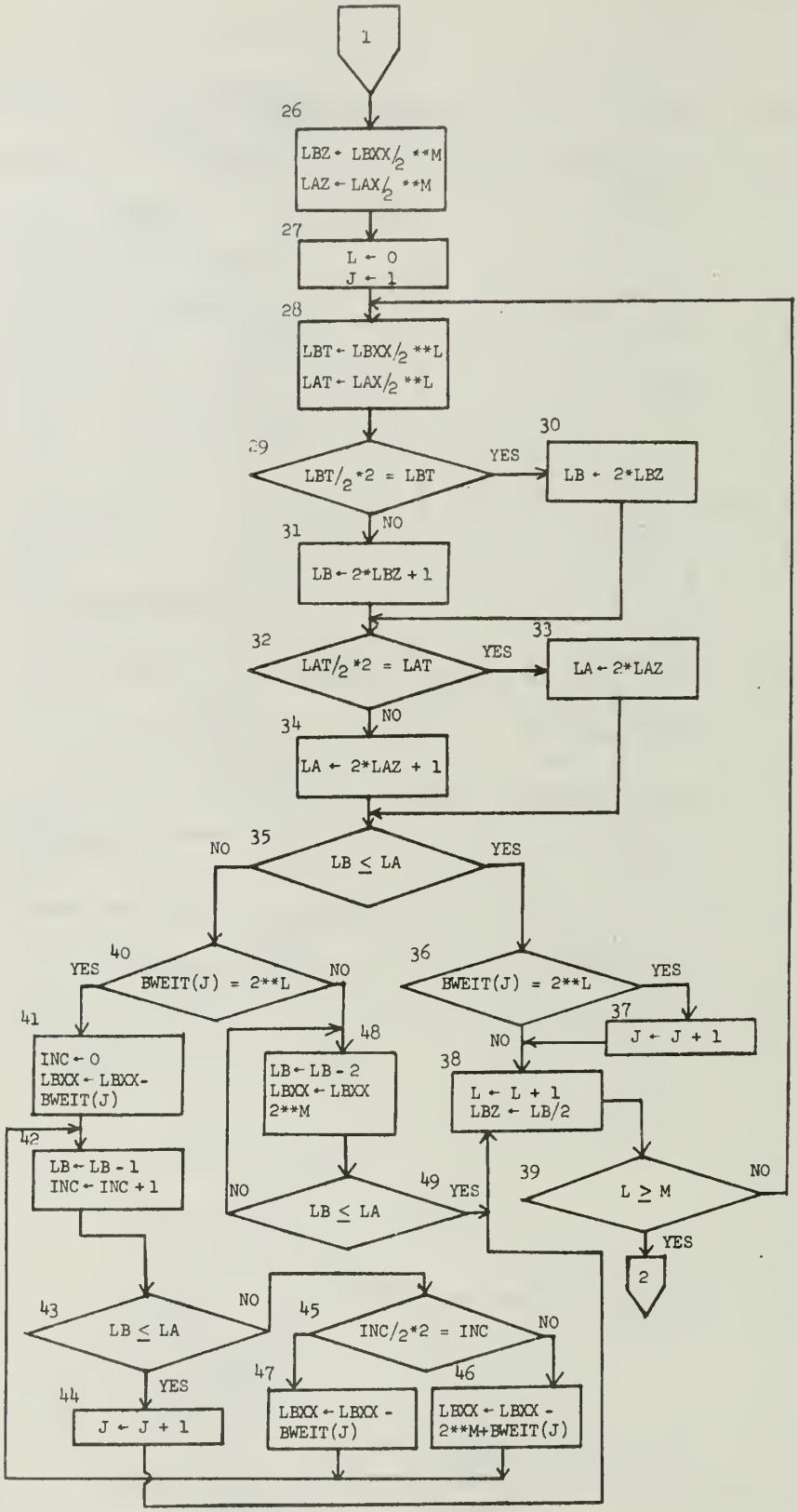


Fig. 3.3.7(b) Flowchart of subroutine CMXL (continued).

LBX: This is a variable where LABEL(PTRB) is stored after 1's are assigned to the unspecified bits in LABEL (PTRB).

LBXX: This is a variable which stores contents of LBX.

LAX: This is a variable which stores the maximum label that has already been assigned to vertex A.

INB: This is a counter for decreasing LBXX (used in the case of FL = 0).

INC: This is a counter for decreasing LB (used in the case of FL = 1).

In blocks 1, 2 and 3 (Fig. 3.3.7(a)), the maximum label is assigned to the vertex with weight zero. Starting from the vertices with weight zero (block 2), a maximum possible label is assigned to every vertex in the N-cube in a manner similar to the case of subroutine CMNL. The loop consisting of blocks 16, 17 and 18 is replaced by the whole flowchart in Fig. 3.3.7(b) when FL is 1. In block 13, PTRA is obtained by subtracting SHIFT from PTRB.

4. INPUT DATA SETUP

Because of the implementation of the design algorithm described in Chapter 2, some modifications in input data setup of Yamamoto's program have been made. Two main differences are in the parameter card and the restriction on problem size. For users' convenience, not only the differences but also the whole detailed format description are presented as follows.

4.1 Input Data Card Format

For each problem, the following two sets of input data cards must be submitted: < parameter card > and < output function card >s.

(a) < parameter card >

This card contains 3 parameters N, M and FL. The meaning of these parameters is explained below.

N

N is the number of external variables of the given switching function. It is specified in columns 1 and 2, and it is right justified.

M

M is the number of output functions. It is specified in columns 5 and 6, and it is right justified.

FL

FL is a logical variable. It is used to indicate which multiple-output case is considered. When FL is specified to 1, it implies that the output functions will be realized at the output level. When FL is

specified to 1, it implies that the output functions will be realized at the output level. When FL is specified to 0, it means that the output functions will be realized at the last M levels. In single-output case, FL can have either 1 or 0. FL is specified in column 9.

Column 3 - column 4, column 7 - column 8 and all the other columns are blank.

(b) < output function card >s

Although output functions are submitted to this program in the truth table form, the input vector for each output function value is implicitly specified by the order in which the function values are specified in each output function card(s). The truth table with N external input variables and M output functions is shown in Fig. 4.1.

Each output function is specified on one or several output function cards, depending on the number of external input variables, N. For N external input variables, there are 2^N possible combinations of input vectors and so the number of components of each output function value is 2^N.

In Fig. 4.1, the first component of the column for the first function f_1 is specified in the first column of the first output function card(s). It is specified to 1, 0, or * if f_1^0 is 1, 0, or a don't care, respectively. f_1^1, the second component of function f_1 is specified in the second column and so forth. After specifying all the function values of function f_1, function f_2 is specified on separate output function card(s). The above process is repeated until M output functions are specified on M separate sets of output function card(s). Since blank column terminates one output function specification (the program DIMN interprets the blank character

x_1	x_2	...	x_{N-1}	x_N	f_1	f_2	- -	- -	f_M
0	0	...	0	0	f_1^0	f_2^0	- -	- -	f_M^0
0	0	...	0	1	f_1^1	f_2^1	- -	- -	f_M^1
0	0	...	1	0	.	.			.
0	0	...	1	1	.	.			.
					.	.			.
					.	.			.
					.	.			.
					.	.			.
1	1	...	1	0	$f_1^{2^N-2}$	$f_2^{2^N-2}$	- -	- -	$f_M^{2^N-2}$
1	1	...	1	1	$f_1^{2^N-1}$	$f_2^{2^N-1}$	- -	- -	$f_M^{2^N-1}$

Fig. 4.1 Truth table with N external input variables and M output functions.

as terminator of one output function specification), the blank character should not be inserted among function specification except at the end of each function specification.

Since each component of an output function value is specified in one column in the output function card(s), the number of cards needed to specify one output function is $\left\lceil \frac{2^N}{80} \right\rceil$, where $\lceil r \rceil$ denotes the smallest integer not smaller than r. Therefore, if the number of external input variables, N, is larger than or equal to 7, then the number of input vectors, 2^N, will exceed the number of columns in one card and consequently two or more output

function cards are required in order to specify one output function. For example, if N = 9, then $2^9 = 512$ and $\left\lceil \frac{2^9}{80} \right\rceil = 7$. This means that we need 7 cards to specify each output function with nine external input variables.

In Fig. 4.2, input data cards for one problem are shown and in Fig. 4.3(a), and input card sequence for the execution of a typical DIMN problem using the batch system of IBM 360 is shown. (Fig. 4.3(b) shows that for Cyber 175.) If there exists a format error in input data, the following error message is printed out and the program execution halts.

"INPUT ERROR IN DATA CARD I = *** J = *** K = ***"

This message means that an error occurred at the K-th column of the J-th output function card which specifies the I-th output function.

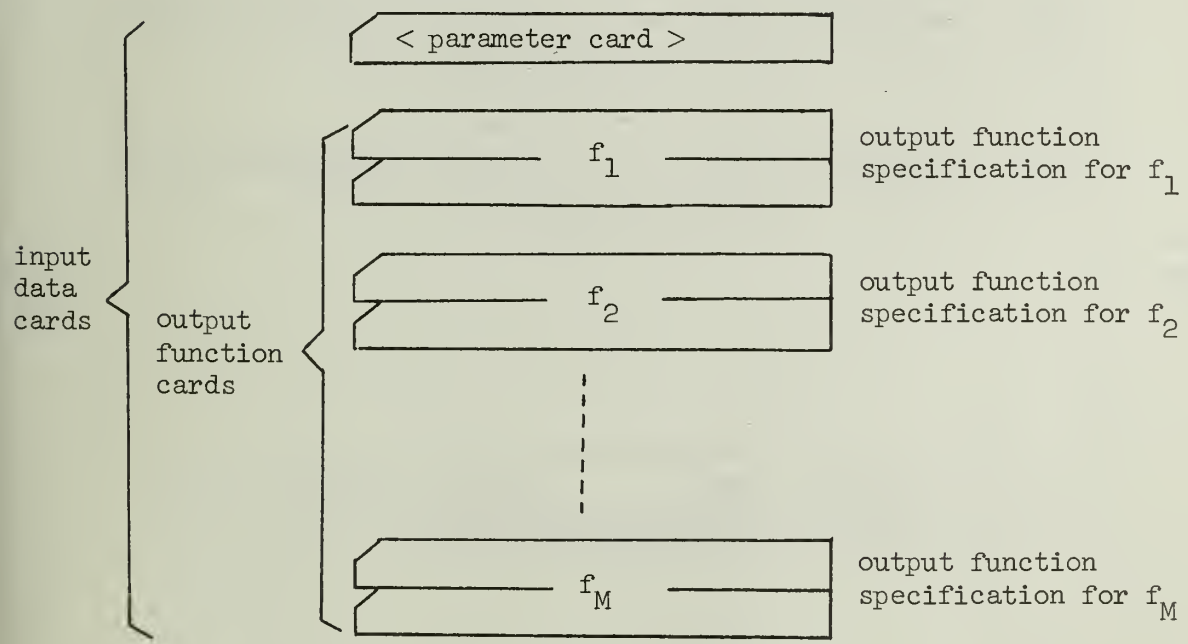
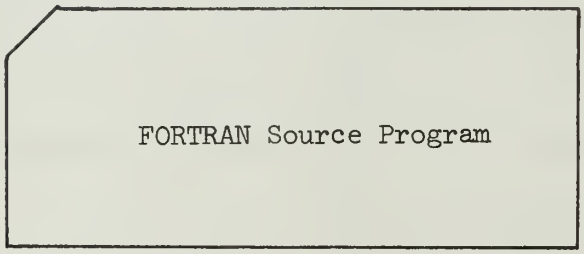
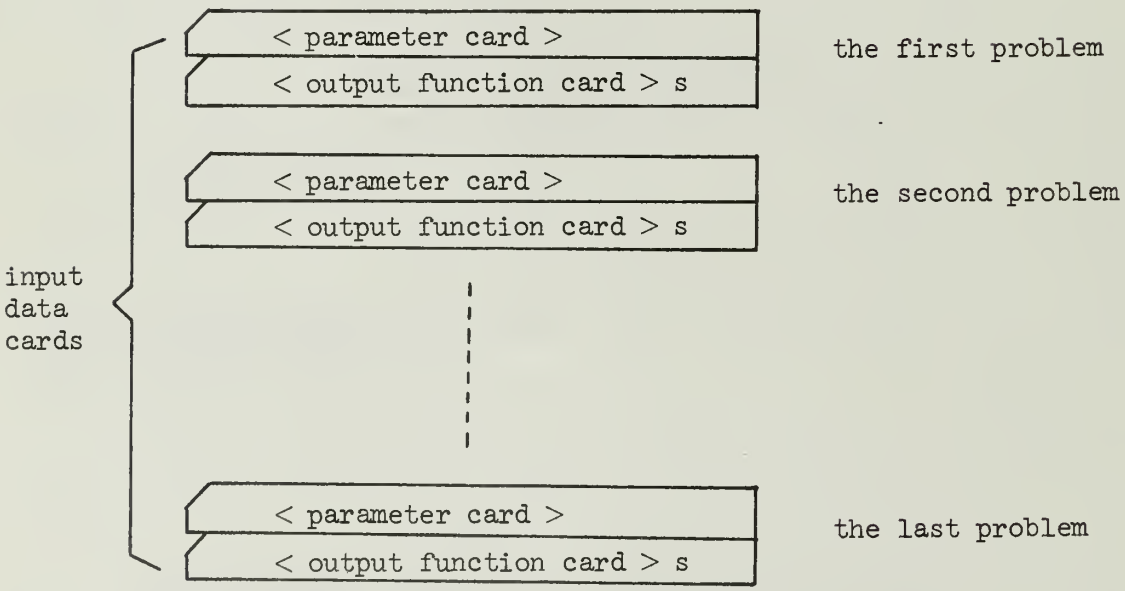


Fig. 4.2 The output data cards for one problem.

```
//  JOB
/*  ID < ID card information >
/*  ID REGION = 300K, TIME = (00,30), LINES = 06000
//  EXEC FORTLDGO, REGION.  GO = 300K
```




```
/*
//  GO.  SYSIN  DD *
```




```
/*
```

Fig. 4.3(a) Input card sequence for the execution of a typical DIMN problem on IBM 360.

JOB.

SIGNON (ID NUMBER)

< PASSWORD >

< CHARGE INFORMATION >

FTN.

LGO.

/7/8/9

FORTRAN Source Program

/7/8/9

input data cards

< parameter card >
< output function card > s — the first problem

< parameter card >
< output function card > s — the second problem

< parameter card >
< output function card > s — the last problem

/6/7/8/9

Fig. 4.3(b) Input card sequence for the execution of a typical DIMN problem on CYBER 175.

4.2 Restriction on Problem Size

Some restrictions on the size of an acceptable problem are required in order to pack problems into a finite amount of space. The size restriction of each problem to be solved by this program DIMN depends on the dimension of the Large-cube. For single-output functions, the dimension of the Large-cube will be $N+i-1$ at the i-th iteration of the program loop, and each time the program loop is executed, the dimension of the Large-cube will be increased by one. Therefore, at the last iteration of this program loop (when $i = R_F$) the dimension of the Large-cube will be $N + R_F - 1$. The restriction on the problem size is due to this memory size. Let $D_I^f(0)$ denote the maximum number of inverse edges over all directed paths from vertex I to vertex 0 in the N-cube. Then, R_F can be represented as follows.

$$R_F = \left\lceil \log_2(D_I^f(0) + 1) \right\rceil + 1 \quad \text{- - - - - - - - - (4.1)}$$

(See Corollary 3.2 in Lai's thesis.)

In single-output case, the maximum value of $D_I^f(0)$ can be $N/2$ or $(N+1)/2$, depending on whether N is even or odd, respectively. Thus $R_F = \left\lceil \log_2(D_I^f(0) + 1) \right\rceil + 1 = \left\lceil \log_2(\frac{N}{2}+1) \right\rceil + 1$ holds if N is even, or $R_F = \left\lceil \log_2(\frac{N+1}{2} + 1) \right\rceil + 1$ holds if N is odd. It follows that the restriction on problem size has to satisfy the following relationships.

$$N + 1 + \left\lceil \log_2(\frac{N}{2} + 1) \right\rceil \leq L \quad \text{if N is even}$$

or

$$N + 1 + \left\lceil \log_2(\frac{N+1}{2} + 1) \right\rceil \leq L \quad \text{if N is odd}$$

where L is the reserved dimension for the corresponding Large-cube.

Based on the above relationships, the maximum number of external variables which the program DIMN can handle is eight (since the space reserved for Large-cube in program DIMN is 4K, i.e., L = 12). This is the size restriction for single-output functions.

Let us consider the cases of multiple-output functions. Obviously it is possible that every directed edge in the N-cube for a multiple-output function is an inverse edge. An example of 3 external input variables and 2 output functions is shown in Fig. 4.4. Therefore, in the worst case of multiple-output functions, $D_I^f(0)$ is equal to the number of

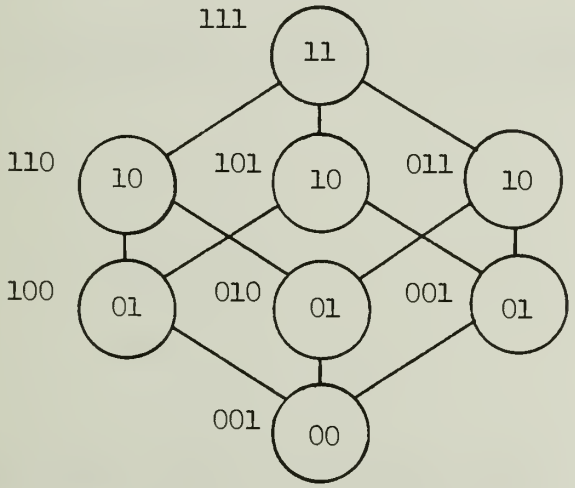


Fig. 4.4 Every directed edge in this 3-cube is an inverse edge.

external input variables, N. The last term in eq. (4.1) represents that only one output gate is required to realize the given single-output function. Therefore, if all the output functions $f_1,\ldots,f_M$ are realized at the last M levels, the last term in eq. (4.1) must be changed to M because the output functions $f_1,\ldots,f_{i-1}$ together with the sequence $x_1,\ldots,x_N,u_1,\ldots,u_{R_F-M}$ are used as input vectors to the corresponding Large-cube to realize function f_i, for i = 1,...,M. Consequently, the

restriction on problem size for multiple-output functions with all output functions realized at the last M levels has to satisfy the following relationship.

$$N + M + \left\lceil \log_2(N+1) \right\rceil \leq L$$

The relationship between the maximum number of external variables and the maximum number of output functions which the program DIMN can handle is shown in Table 4.1 assuming that L is 12.

maximum number of external variables, N	maximum number of output functions, M
7	2
6	3
5	4
4	5
3	7
2	8

Table 4.1 Size restriction for multiple-output functions with all output functions realized at the last M levels that program DIMN can handle.

If all the output functions $f_1,\ldots,f_M$ are realized at the last level, the number of negative gates required to realize the desired MOS network, i.e., R_F, can be expressed by eq. (4.1) where $D_I^f(0)$ is replaced by N (as explained previously). The reason is that gates which realize output functions do not feed other gates in the network. While realizing output functions f_i for i = 1,...,M, only the sequence $x_1,\ldots,x_N,u_1,\ldots,u_r$ are used as input vectors to the corresponding Large-cube. By Step 6 of the

Modified Algorithm DIMN, all output functions f_i for i = 1,...,M are realized as M single-output functions with respect to external inputs $x_1,\ldots,x_N,u_1,\ldots,u_r$. Consequently, the restriction on problem size for multiple-output functions with all the output functions realized at the last level has to satisfy the following relationship:

$$N + 1 + \left\lceil \log_2(N+1) \right\rceil \leq L \ .$$

Based on this relationship, the maximum number of external input variables that the program DIMN can handle for multiple-output functions with all the output functions realized at the last level is seven.

If the above restriction is violated, the following error message is printed and the program execution halts.

"INPUT ERROR IN PARAMETER CARD N = *** M = *** FL = ***"

In this error message, N is the number of external variables, M is the number of output functions, and FL is the flag which indicates which case of multiple-output functions is being solved. These limitations are essentially imposed by the array sizes in the programs presently written. The restriction can be loosened by increasing array dimension appropriately. The above restriction on problem size is derived based on the worst-case assumption for each case. Actually, the size limitation depends on each particular problem.

Example 4.1 The truth table for this example function is shown in Fig. 4.5. The input data setup for the network with three external variables (N=3) and two completely specified output functions (M=2) is shown in Fig. 4.6. The output functions are realized at the last 2 levels (FL=0).

Example 4.2 The truth table for the functions is shown in Fig. 4.7. The input data setup for the network with three external variables (N=3) and two incompletely specified output functions (M=2) is shown in Fig. 4.8. The output functions are realized at the last level (FL=1).

x_1	x_2	x_3	f_1	f_2
0	0	0	0	1
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

Fig. 4.5 Truth table for Example 4.1.

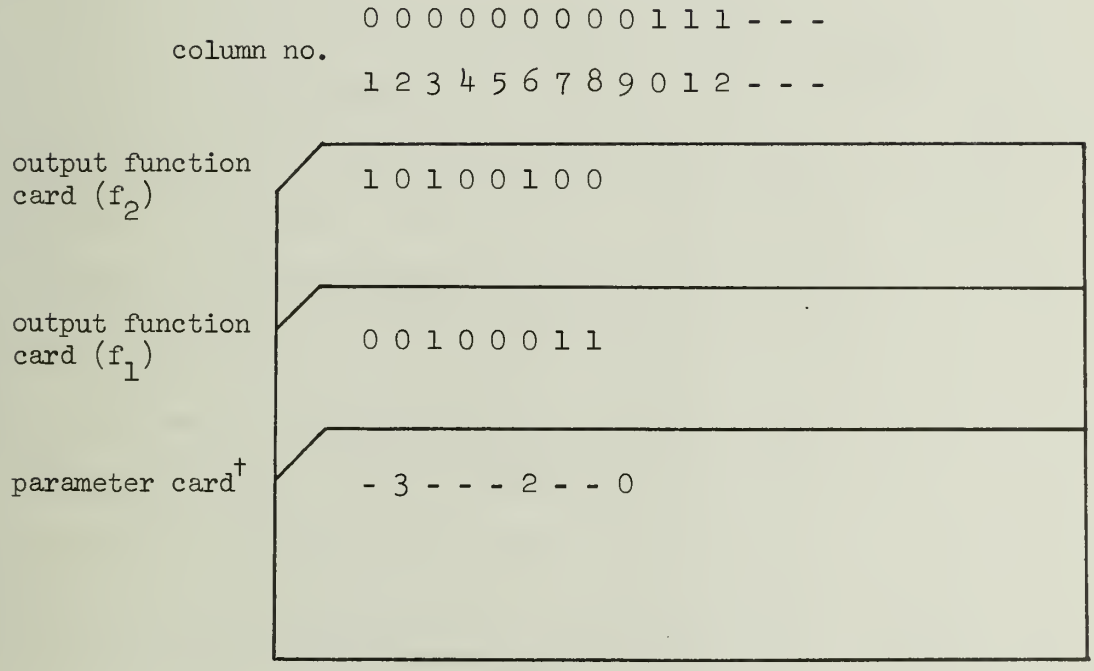


Fig. 4.6 Possible setup of data cards to specify the problem given in Example 4.1.

†: - represents the blank character.

x_1	x_2	x_3	f_1	f_2
0	0	0	1	*
0	0	1	*	0
0	1	0	0	1
0	1	1	1	*
1	0	0	*	0
1	0	1	*	*
1	1	0	0	1
1	1	1	1	1

Fig. 4.7 Truth table for Example 4.2.

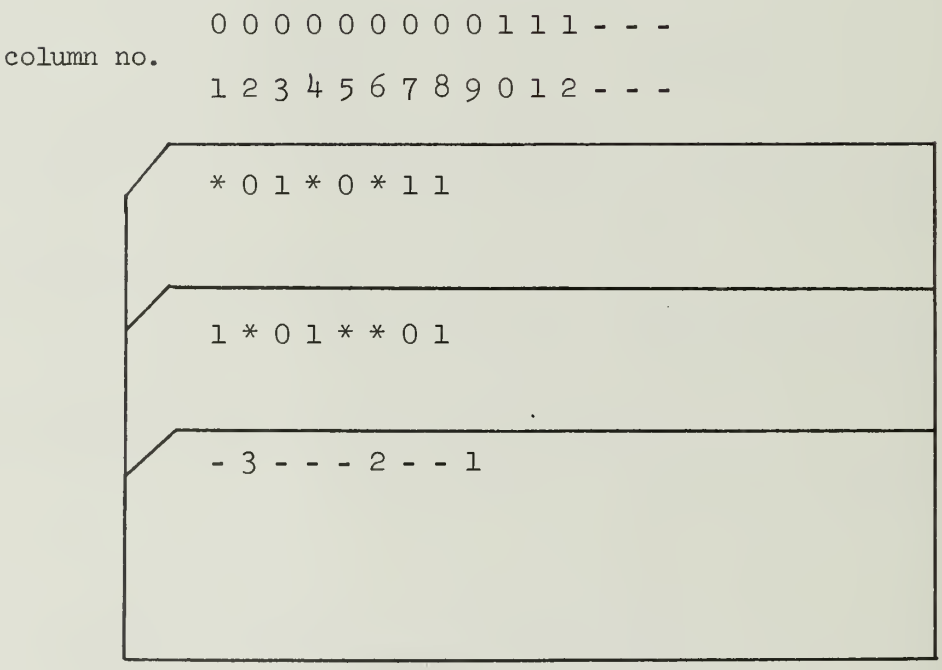


Fig. 4.8 Possible setup of data cards to specify the problem given in Example 4.2

5. OUTPUT OF PROCEUDRE DIMN

In this chapter, the output of program DIMN is described. The output format for a typical DIMN problem is shown in Section 5.1. Some MOS networks obtained by program DIMN are compared for both restrictions on output positions in Section 5.2.

5.1 Output Format

Fig. 5.1 and Fig. 5.3 show the printouts obtained by program DIMN for two typical examples.

In Fig. 5.1, the number of external variables and the number of output functions are printed first. Then the restriction on output cell positions and the two output functions in truth table form are shown. The MOS network configuration for each cell is then printed. Here each variable $x_1,x_2,\ldots,x_N$ represents a driver MOSFET which is fed by the corresponding external variable $x_1,x_2,\ldots,x_N$ and each variable $U_1,U_2,\ldots,U_R$ represents a driver MOSFET which is fed by the corresponding output of cell $g_1,g_2,\ldots,g_R$. In this example, since the given output functions are realized at the last level, MOS cell 3 (or g_3) and MOS cell 4 (or g_4) realize output functions f_1 and f_2, respectively, and both cells are at the last level. Finally the number of MOS cells, the number of driver MOSFET's and the time elapsed† are shown one by one. Fig. 5.2 shows the network obtained from the MOS cell configuration printed in Fig. 5.1.

---

† For the IBM 360/75J computer, STEPZ in FORTRAN system subroutines is used as a timing subroutine. In addition, program DIMN is compiled by FORTRAN G compiler.

```
*****************************************
*                                       *
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*                                       *
*****************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

*****************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST LEVEL

FUNCTION  1
01101001

FUNCTION  2
00010111

NETWORK CONFIGURATION

                      |--X2--X3--|
                      |          |
MOS CELL 1   0--------|--X1--X3--|--0
                      |          |
                      |--X1--X2--|

                      |--X1--X2--X3--|
                      |              |
                      |--X3--U1------|
MOS CELL 2   0--------|              |--0
                      |--X2--U1------|
                      |              |
                      |--X1--U1------|

MOS CELL 3   0-----U2-----0

MOS CELL 4   0-----U1-----0

NUMBER OF MOS CELLS =  4
NUMBER OF MOS FETS  = 17
(WITHOUT FACTORING)

ELAPSED TIME =    0.12 SEC
```

Fig. 5.1 Printout for given functions is realized in output level.

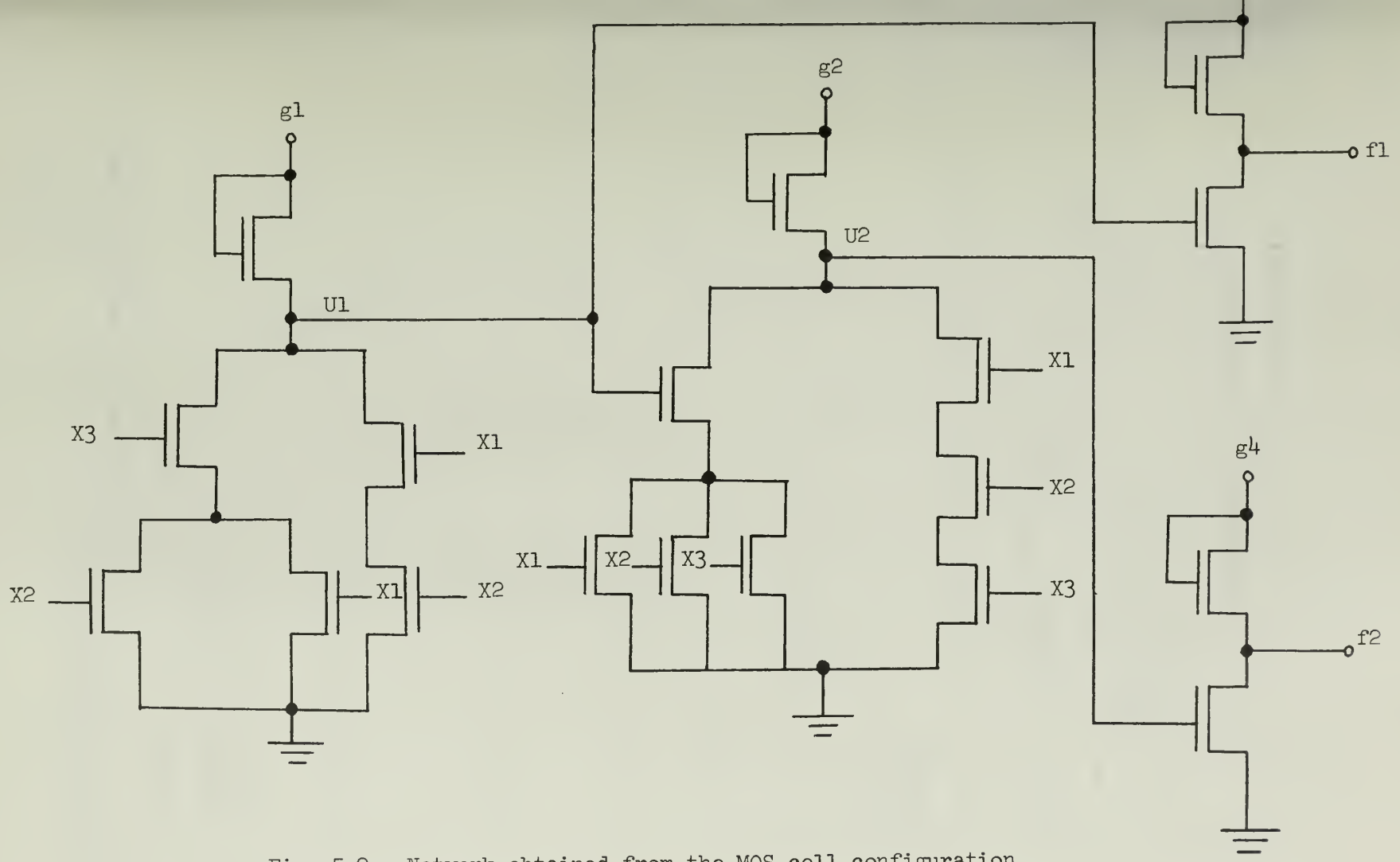


Fig. 5.2 Network obtained from the MOS cell configuration in Fig. 5.1.

```
*****************************************
*                                       *
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*                                       *
*****************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

*****************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST  2 CELLS

FUNCTION  1
01101001

FUNCTION  2
00010111

NETWORK CONFIGURATION

MOS CELL 1  0-----X3-----0

                  |--X2--U1--|
                  |          |
MOS CELL 2  0--|--X1--U1--|--0
                  |          |
                  |--X1--X2--|

                  |--X1--X2--U1--|
                  |              |
                  |--U1--U2------|
MOS CELL 3  0--|                 |--0
                  |--X2--U2------|
                  |              |
                  |--X1--U2------|

                  |--U2--U3--|
                  |          |
MOS CELL 4  0--|--U1--U3--|--0
                  |          |
                  |--U1--U2--|

NUMBER OF MOS CELLS =  4
NUMBER OF MOS FETS  = 22
(WITHOUT FACTORING)

ELAPSED TIME =   0.17 SEC
```

Fig. 5.3 Printout for given functions is realized in last 2 levels.

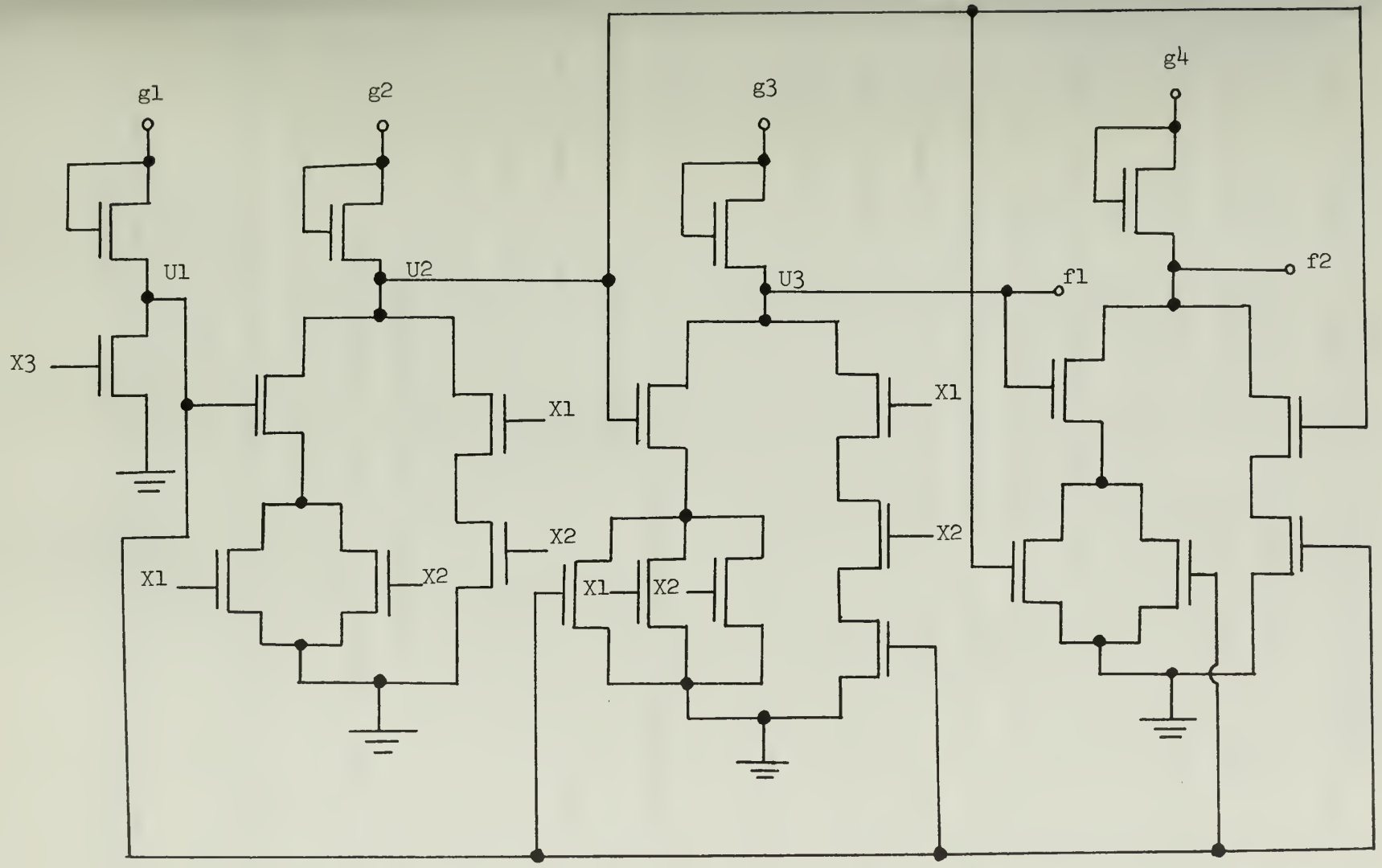


Fig. 5.4 Network obtained from the MOS cell configuration in Fig. 5.3.

Fig. 5.3 shows the MOS network with the given output functions realized at the last 2 levels. Therefore, MOS cell 3 (or g_3) and MOS cell 4 (or g_4) realize output functions f_1 and f_2, respectively. In this example, the output function f_1 realized by MOS cell 3 feeds to MOS cell 4. Each function u_i for $i = 1,\ldots,R_F$, realized by MOS cell g_i, respectively, is in the sum-of-products (disjunctive) form. Sometimes the number of FET's in an MOS cell designed by Lai's algorithm (though these FET's are irredundant) can be reduced by factoring out common literals in the expression of u_i. Obviously, this factorization does not change the irredundancy of the MOS cell. The network information obtained in Fig. 5.3 is redrawn in Fig. 5.4. In both networks, factoring of literals in switching expressions in each MOS cell is done by hand.

5.2 Networks Obtained by Program DIMN

In this section, based on several example functions† some comparisons are made between networks with all the output functions realized at the last level and networks with all the output functions realized at the last M levels. As mentioned in Chapter 2, the network configurations obtained in the latter case depend on the permutations of the output cell positions of output functions. Thus, the comparison shown in Table 5.2.1 includes both the best and the worst results which are obtained from different output cell positions in the latter case. Here, by "best result,"

---

† More than 20 three and four variable functions have been tested by program DIMN, but only few examples are shown in Table 5.2.1.

Table 5.2.1 Some comparisons for multiple-output functions obtained by program DIMN.

No. of variables	No. of output functions	cases *		Functions	No. of MOS cells	No. of driver FETs **	No. of interconnections
3	2	FL = 1		1-bit full adder	4	17	5
		FL = 0	W	"	4	22	13
			B	"	4	17	5
5	3	FL = 1		2-bit full adder	6	158	61
		FL = 0	W	"	6	166	72
			B	"	6	106	49
3	2	FL = 1		f_1 = B9, f_2 = 1E	4	13	4
		FL = 0	W	"	4	16	7
			B	"	3	15	4
3	2	FL = 1		f_1 = 3A, f_2 = 2B	4	12	6
		FL = 0	W	"	4	14	8
			B	"	4	10	4
3	2	FL = 1		f_1 = 4C, f_2 = 8A	3	7	2
		FL = 0	W	"	3	7	3
			B	"	3	7	2
3	2	FL = 1		f_1 = 95, f_2 = 27	4	14	6
		FL = 0	W	"	4	14	8
			B	"	4	14	6
3	2	FL = 1		f_1 = B3, f_2 = 42	4	15	5
		FL = 0	W	"	4	13	5
			B	"	3	14	5

* FL = 1 indicates the designed network with all the output functions realized at the last level.
FL = 0 indicates the designed network with all the output functions realized at the last M levels.
W represents the worst result for the case of FL = 0.
B represents the best result for the case of FL = 0.

** without factoring

we mean that the number of MOS cells in the network obtained is the smallest among all networks corresponding to all output cell positions. Similarly with "worst result". In particular, in the case that all networks have the same number of MOS cells, the best result is the network that has the smallest number of driver MOSFETs and the worst result is the network that has the largest number of driver MOSFETs. In the case that all networks have the same number of MOS cells and the same number of driver MOSFETs, the best result is the network that has the smallest number of interconnections and the worst result is the network that has the largest number of interconnections.

In this table, functions are represented in hexadecimal form. For example, function f = 01001100 is represented in hexadecimal form f = 4C. As can be seen in this table, the results obtained for networks with output functions $f_1,\ldots,f_M$ realized at the last M levels extremely depend on the permutation of output functions. If a permutation is chosen inappropriately, not only the number of MOS cells but also the number of driver FETs or (and) the number of interconnections obtained may be greater than those obtained by other permutations. However, those permutations from which the best results are obtained are usually not possible to be found unless we exhaust all the possible permutations. Although different permutations (for a network with M output functions, there are M! permutations) may result in different MOS networks, each network designed by Lai's algorithm is guaranteed to be irredundant.

For a given set of functions, only networks with all output functions to be realized at the last level or networks with all output function to

REFERENCES

[1] T. Ibaraki and S. Muroga, "Synthesis of networks with a minimum number of negative gates," IEEE Trans. Computers, Vol. C-20, pp. 49-58, Jan. 1971.

[2] K. Nakamura, N. Tokura, and T. Kasami, "Minimal negative gate networks," IEEE Trans. Computer, Vol. C-21, pp. 5-11, Jan. 1972.

[3] T. K. Liu, "Synthesis of logic networks with MOS complex cells," Report No. UIUCDCS-R-72-517, Department of Computer Science, University of Illinois, Urbana, Illinois, May 1972.

[4] H. C. Lai, "Design of diagnosable networks," Ph.D. thesis, 1976 Department of Computer Science, University of Illinois, Urbana, Illinois.

[5] K. Yamamoto, "Design of irredundant MOS networks: A program manual for the design Algorithm DIMN," Report No. UIUCDCS-R-76-784, Department of Computer Science, University of Illinois.

[6] S. Muroga, CS 363 - "Integrated Circuit Logical Design," classnotes, Department of Computer Science, University of Illinois, Urbana, Illinois.

APPENDIX A

Networks Obtained by Program DIMN

REFERENCES

[1] T. Ibaraki and S. Muroga, "Synthesis of networks with a minimum number of negative gates," IEEE Trans. Computers, Vol. C-20, pp. 49-58, Jan. 1971.

[2] K. Nakamura, N. Tokura, and T. Kasami, "Minimal negative gate networks," IEEE Trans. Computer, Vol. C-21, pp. 5-11, Jan. 1972.

[3] T. K. Liu, "Synthesis of logic networks with MOS complex cells," Report No. UIUCDCS-R-72-517, Department of Computer Science, University of Illinois, Urbana, Illinois, May 1972.

[4] H. C. Lai, "Design of diagnosable networks," Ph.D. thesis, 1976 Department of Computer Science, University of Illinois, Urbana, Illinois.

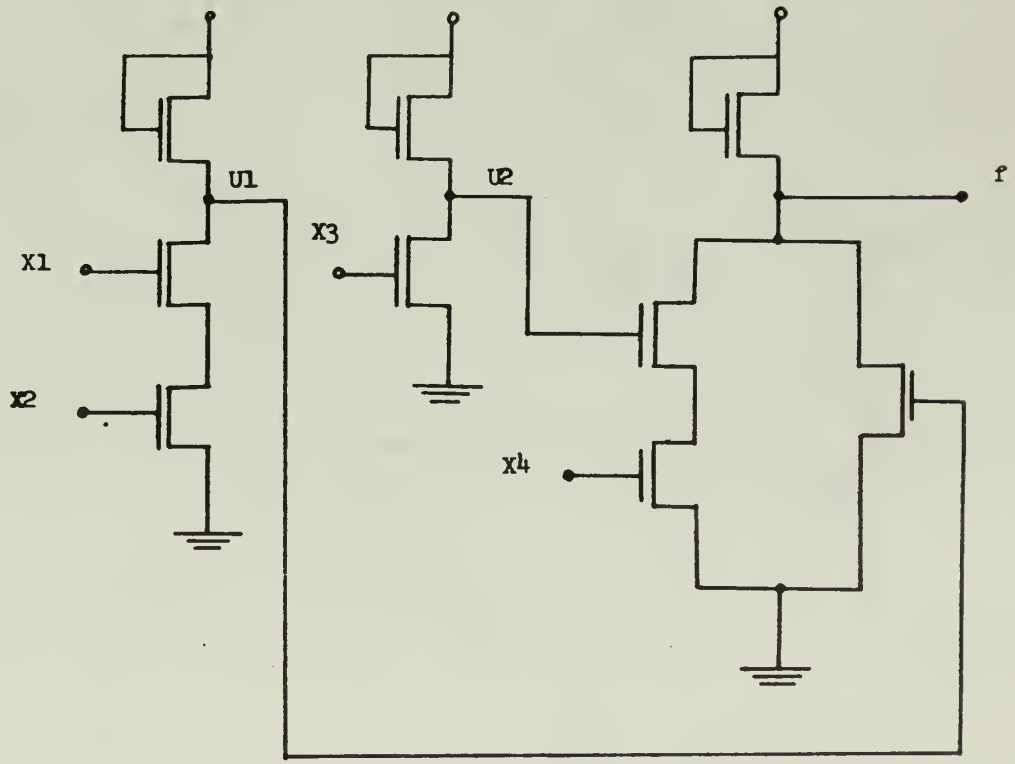
[5] K. Yamamoto, "Design of irredundant MOS networks: A program manual for the design Algorithm DIMN," Report No. UIUCDCS-R-76-784, Department of Computer Science, University of Illinois.

[6] S. Muroga, CS 363 - "Integrated Circuit Logical Design," classnotes, Department of Computer Science, University of Illinois, Urbana, Illinois.

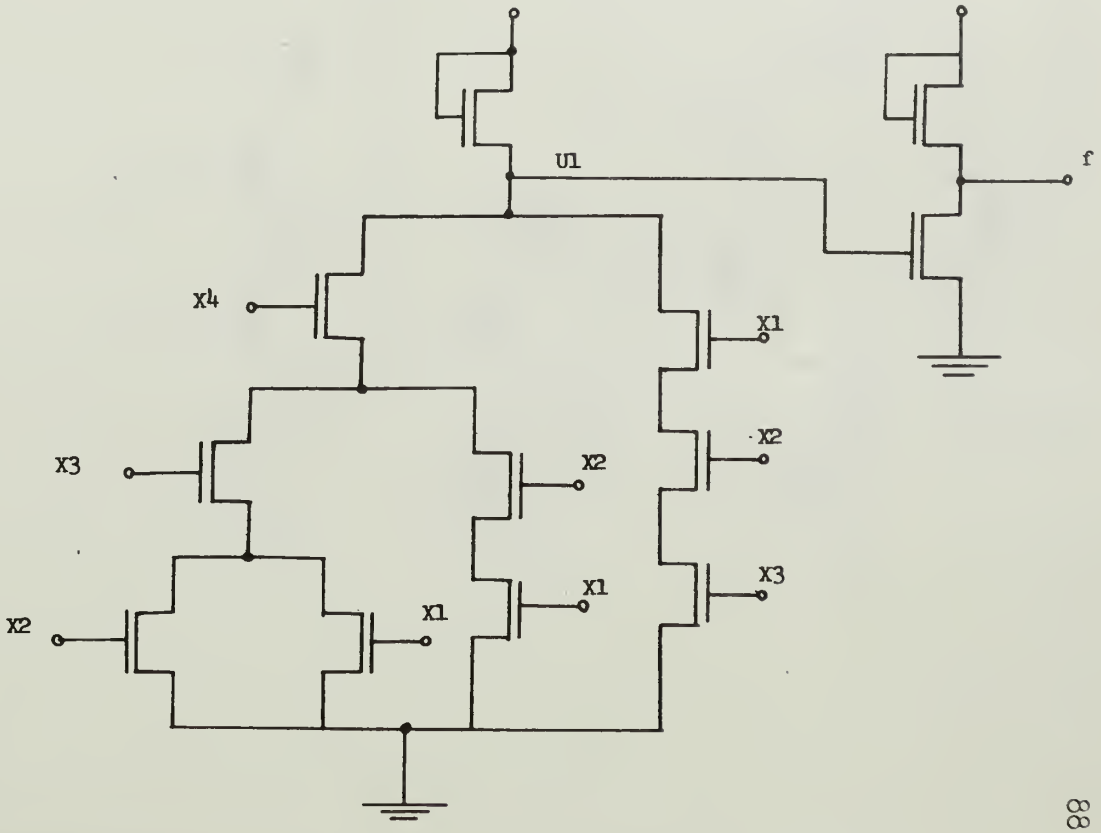
APPENDIX A

Networks Obtained by Program DIMN

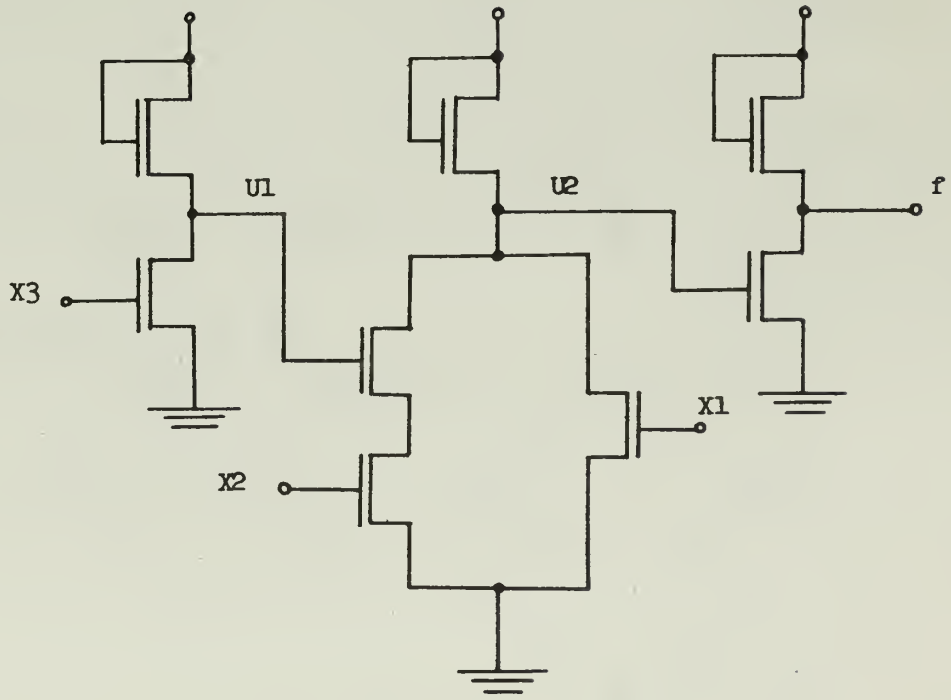
```
**********************************************
*                                            *
*   DESIGN OF IRREDUNDANT MOS NETWORK        *
*                                            *
**********************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

**********************************************

NUMBER OF EXTERNAL VARIABLES =  4
NUMBER OF OUTPUT FUNCTIONS   =  1

FUNCTION  1
0000000000001011

NETWORK CONFIGURATION

MOS CELL 1   0-----X1--X2-----0

MOS CELL 2   0-----X3-----0

MOS CELL 3   0--|--X4--U2--|--0
                |--U1------|

NUMBER OF MOS CELLS =  3
NUMBER OF MOS FETS  =  6
(WITHOUT FACTORING)

ELAPSED TIME =   0.15 SEC
```



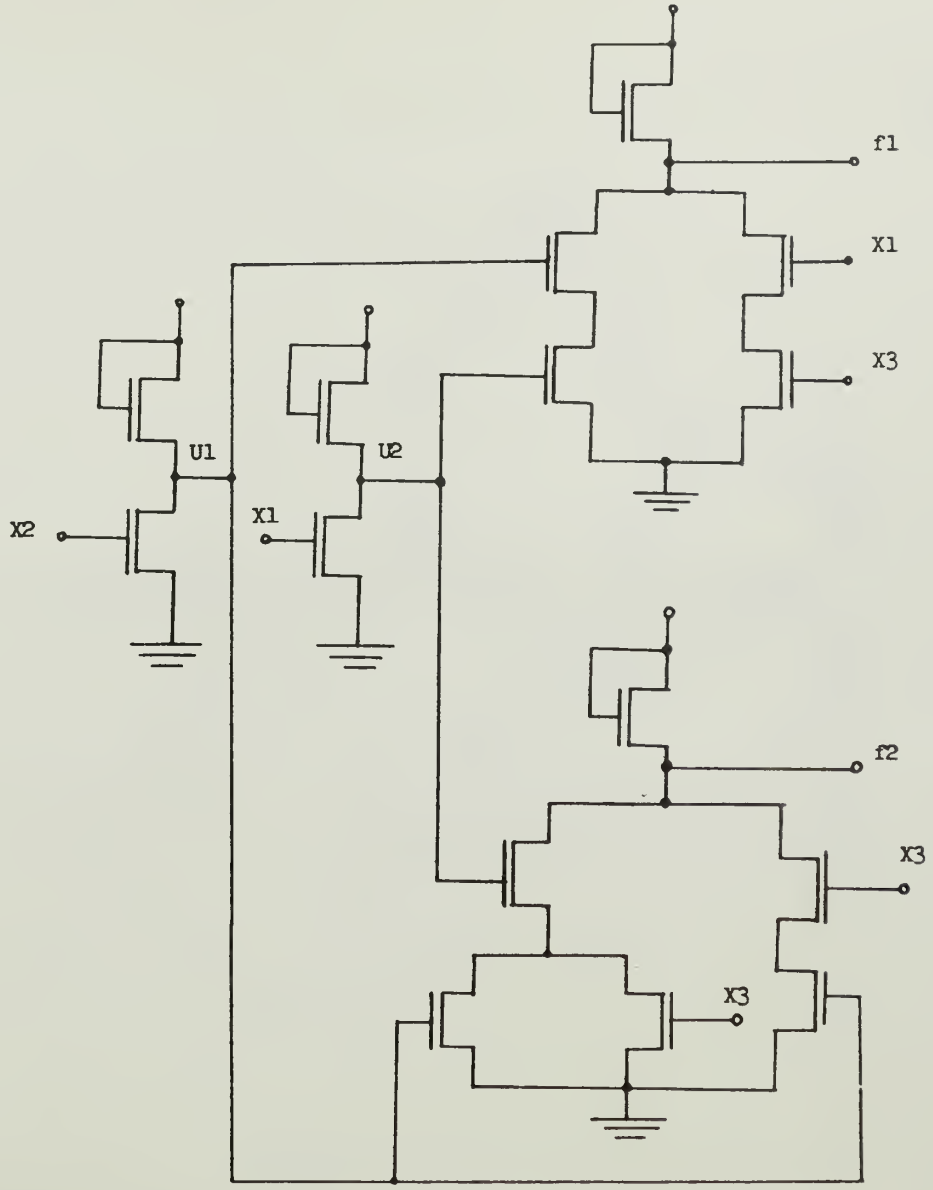
```
*****************************************
*                                       *
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*                                       *
*****************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

*****************************************

NUMBER OF EXTERNAL VARIABLES =  4
NUMBER OF OUTPUT FUNCTIONS   =  1

FUNCTION  1
0000000100010111

NETWORK CONFIGURATION

                  |--X2--X3--X4--|
                  |--X1--X3--X4--|
MOS CELL 1  0--   |--X1--X2--X4--|  --0
                  |--X1--X2--X3--|

MOS CELL 2  0-----U1-----0

NUMBER OF MOS CELLS =  2
NUMBER OF MOS FETS  = 13
(WITHOUT FACTORING)

ELAPSED TIME =   0.09 SEC
```



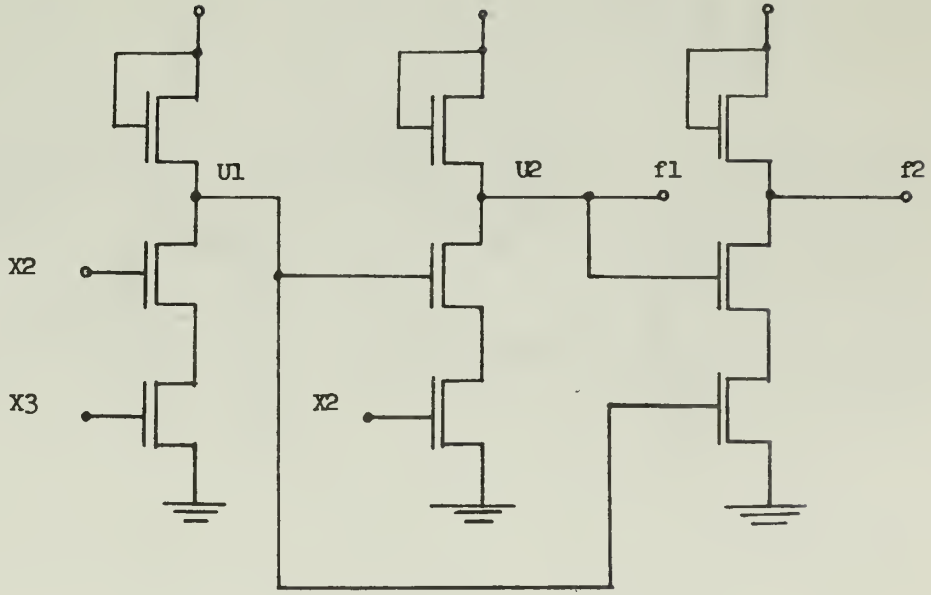
```
*****************************************
*                                       *
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*                                       *
*****************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

*****************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  1

FUNCTION  1
0*10***1

NETWORK CONFIGURATION

MOS CELL 1  0-----X3-----0

MOS CELL 2  0--|--X2--U1--|--0
               |--X1------|

MOS CELL 3  0-----U2-----0

NUMBER OF MOS CELLS =  3
NUMBER OF MOS FETS  =  5
(WITHOUT FACTORING)

ELAPSED TIME =   0.07 SEC
```



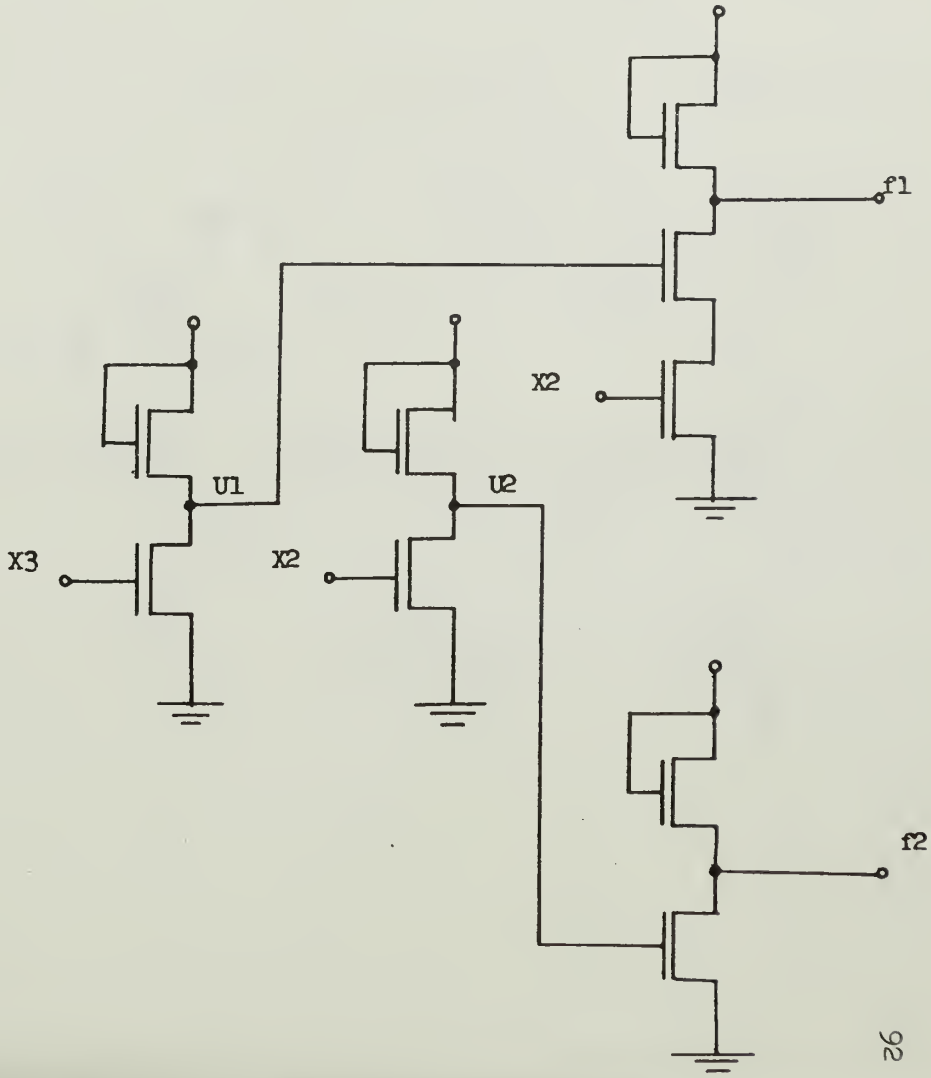
```
**********************************************
*                                            *
*   DESIGN OF IRREDUNDANT MOS NETWORK        *
*                                            *
**********************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

**********************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST LEVEL

FUNCTION  1
00111010

FUNCTION  2
00101011

NETWORK CONFIGURATION

MOS CELL 1  0-----X2-----0

MOS CELL 2  0-----X1-----0

                    |--U1--U2--|
MOS CELL 3  0--     |          |--0
                    |--X1--X3--|

                    |--U1--U2--|
                    |          |
MOS CELL 4  0--     |--X3--U2--|--0
                    |          |
                    |--X3--U1--|

NUMBER OF MOS CELLS =  4
NUMBER OF MOS FETS  = 12
(WITHOUT FACTORING)

ELAPSED TIME =   0.15 SEC
```



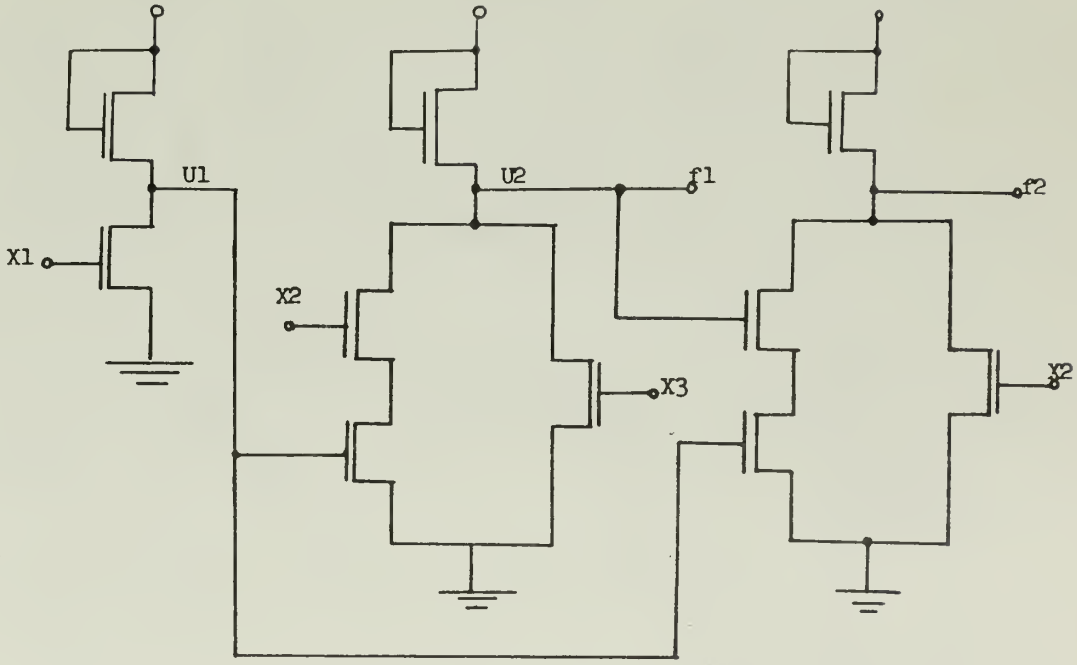
```
*****************************************
*                                       *
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*                                       *
*****************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

*****************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST  2 CELLS

FUNCTION  1
1*01**01

FUNCTION  2
*01*0*11

NETWORK CONFIGURATION

MOS CELL 1  0-----X2--X3-----0

MOS CELL 2  0-----X2--U1-----0

MOS CELL 3  0-----U1--U2-----0

NUMBER OF MOS CELLS =  3
NUMBER OF MOS FETS  =  6
(WITHOUT FACTORING)

ELAPSED TIME =   0.07 SEC
```



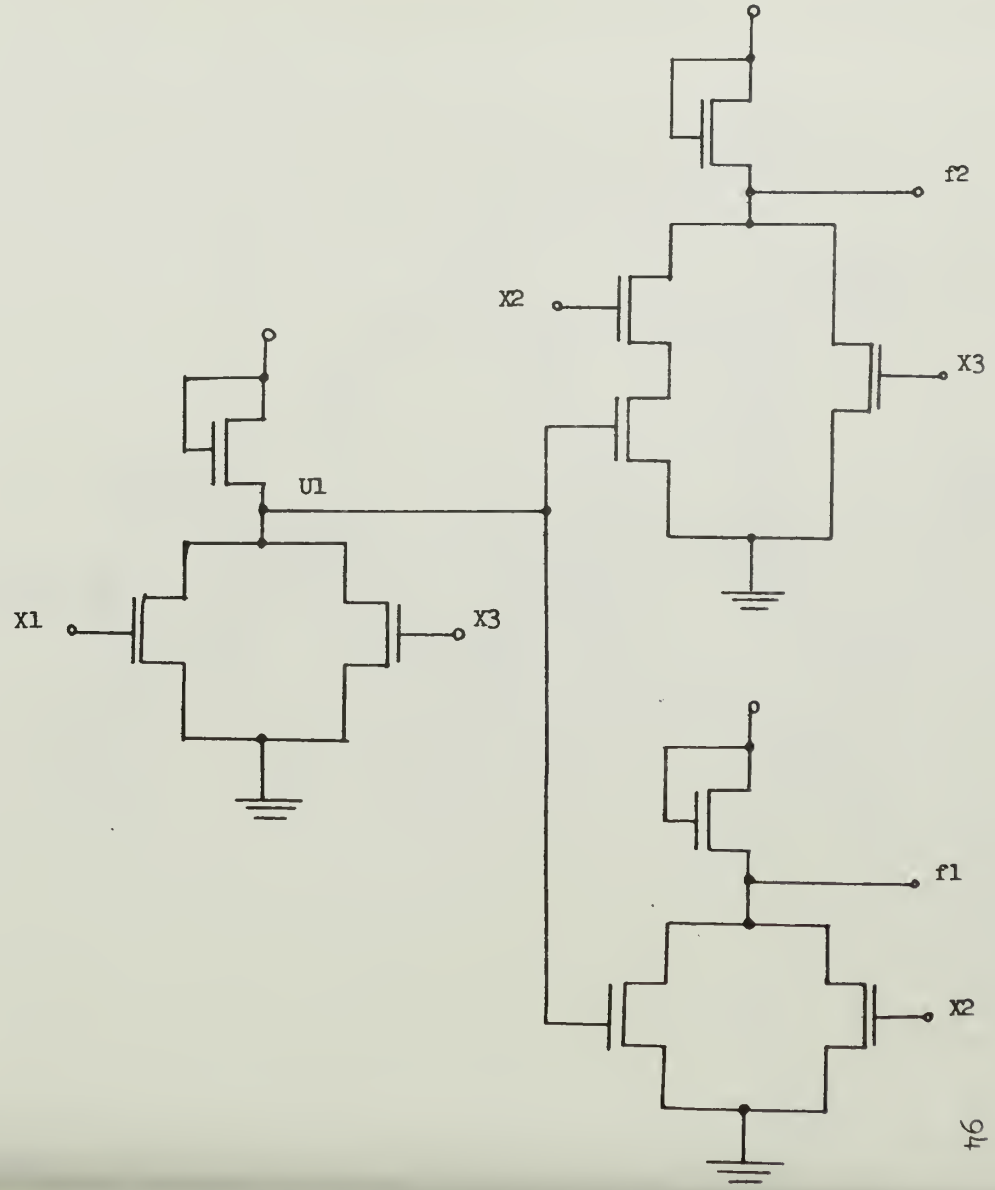
```
*****************************************
*                                       *
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*                                       *
*****************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL
*****************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST LEVEL

FUNCTION  1
1*01**01

FUNCTION  2
*01*0*11

NETWORK CONFIGURATION

MOS CELL 1  0-----X3-----0

MOS CELL 2  0-----X2-----0

MOS CELL 3 .0-----X2--U1-----0

MOS CELL 4  0-----U2-----0

NUMBER OF MOS CELLS =  4
NUMBER OF MOS FETS  =  5
(WITHOUT FACTORING)

ELAPSED TIME =   0.12 SEC
```



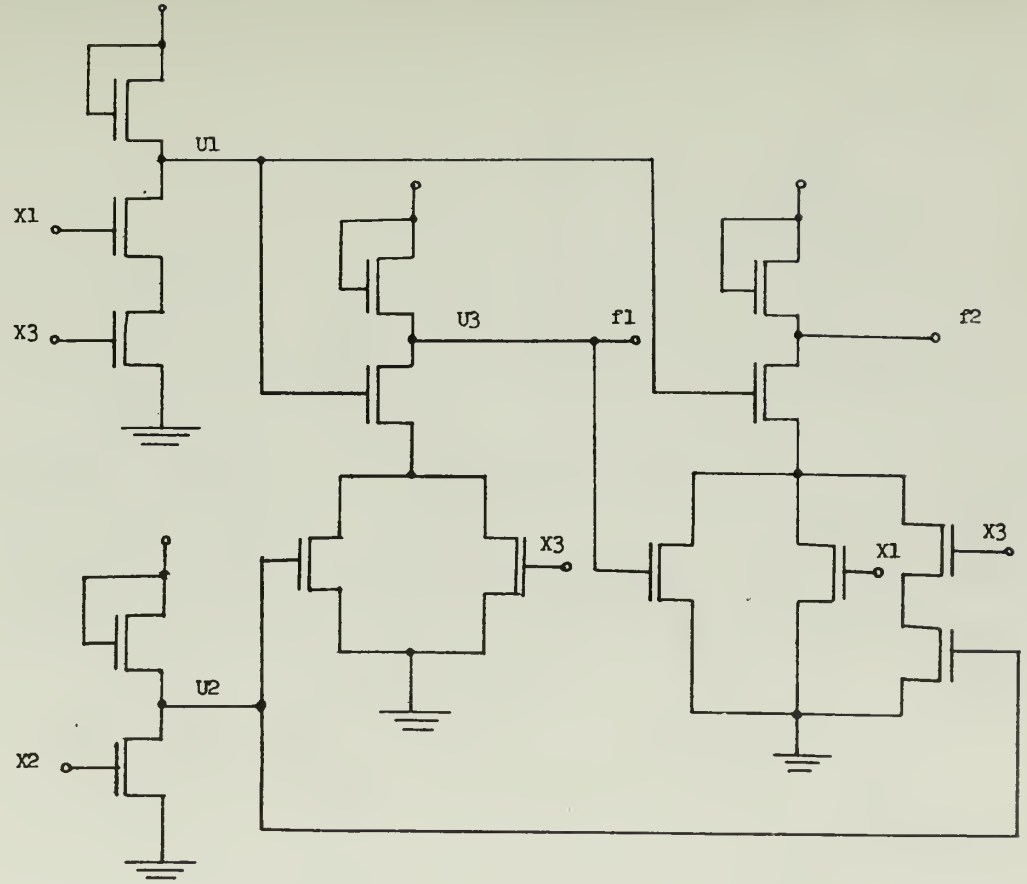
```
**********************************************
*                                            *
*   DESIGN OF IRREDUNDANT MOS NETWORK        *
*                                            *
**********************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

**********************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST  2 CELLS

FUNCTION  1
10007010

FUNCTION  2
01007100

NETWORK CONFIGURATION

MOS CELL 1  0-----X1-----0

MOS CELL 2  0--|--X2--U1--|--0
               |--X3------|

MOS CELL 3  0--|--U1--U2--|--0
               |--X2------|

NUMBER OF MOS CELLS =  3
NUMBER OF MOS FETS  =  7
(WITHOUT FACTORING)

ELAPSED TIME =   0.07 SEC
```



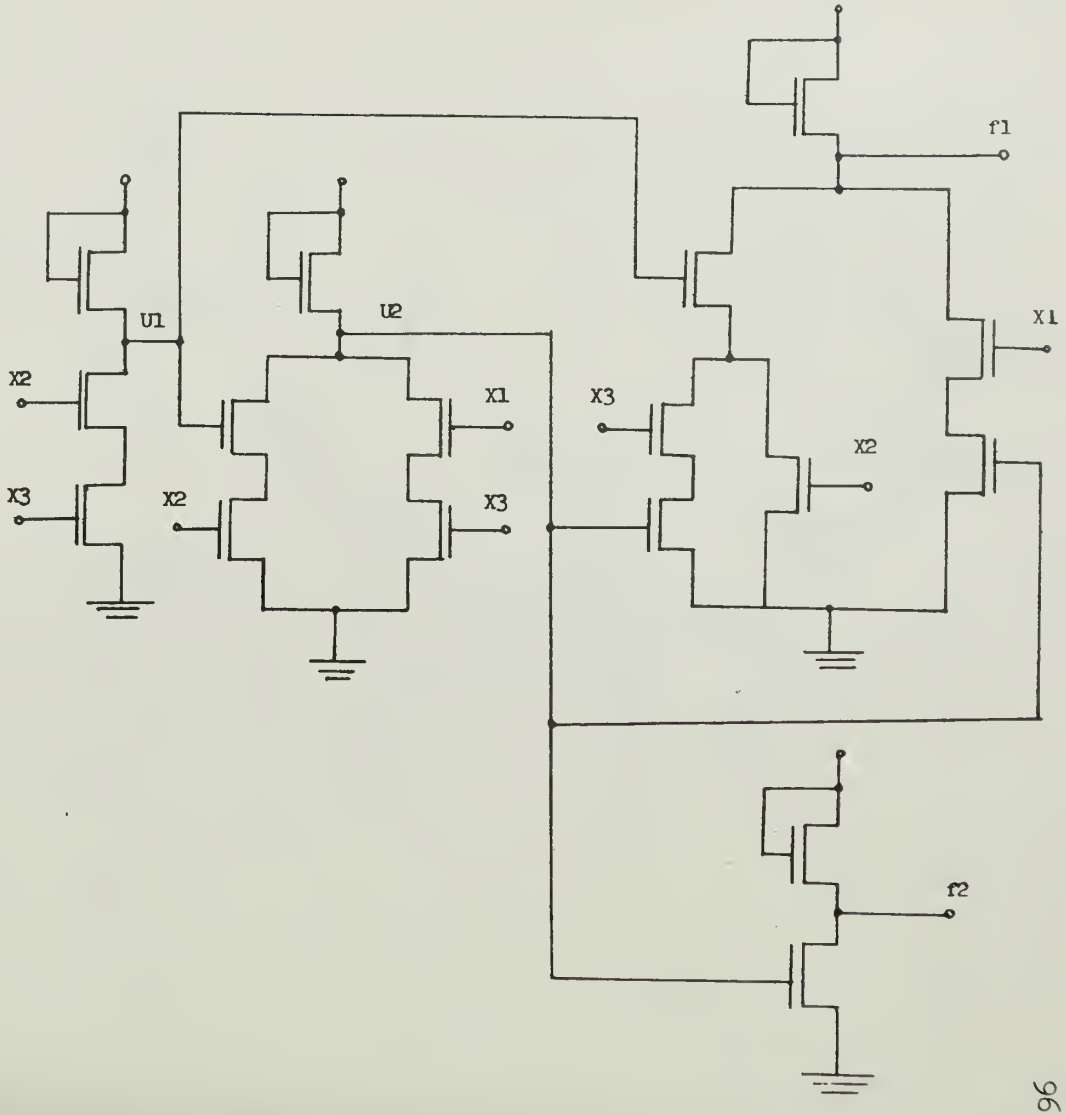
```
**********************************************
*                                            *
*   DESIGN OF IRREDUNDANT MOS NETWORK        *
*                                            *
**********************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

**********************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST LEVEL

FUNCTION  1
01001100

FUNCTION  2
10001010

NETWORK CONFIGURATION

MOS CELL 1   0--|--X3--|--0
                |--X1--|

MOS CELL 2   0--|--U1--|--0
                |--X2--|

MOS CELL 3   0--|--X2--U1--|--0
                |--X3------|

NUMBER OF MOS CELLS =  3
NUMBER OF MOS FETS  =  7
(WITHOUT FACTORING)

ELAPSED TIME =   0.07 SEC
```



```
**************************************
*                                    *
*   DESIGN OF IRREDUNDANT MOS NETWORK*
*                                    *
**************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL
**************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST  2 CELLS

FUNCTION  1
00100111

FUNCTION  2
10010101

NETWORK CONFIGURATION

MOS CELL 1   0-----X1--X3-----0

MOS CELL 2   0-----X2-----0

                  |--U1--U2--|
MOS CELL 3   0--  |          |--0
                  |--X3--U1--|

                  |--X3--U1--U2--|
MOS CELL 4   0--  |--U1--U3------|--0
                  |--X1--U1------|

NUMBER OF MOS CELLS =  4
NUMBER OF MOS FETS  = 14
(WITHOUT FACTORING)

ELAPSED TIME =   0.12 SEC
```



```
**************************************
*                                    *
*   DESIGN OF IRREDUNDANT MOS NETWORK *
*                                    *
**************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

**************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST LEVEL

FUNCTION  1
10010101

FUNCTION  2
00100111

NETWORK CONFIGURATION

MOS CELL 1  0-----X2--X3-----0

MOS CELL 2  0--|--X2--U1--|--0
               |--X1--X3--|

               |--X3--U1--U2--|
               |              |
MOS CELL 3  0--|--X2--U1------|--0
               |              |
               |--X1--U2------|

MOS CELL 4  0-----U2-----0

NUMBER OF MOS CELLS =  4
NUMBER OF MOS FETS  = 14
(WITHOUT FACTORING)

ELAPSED TIME =   0.12 SEC
```



```
**********************************************
*                                            *
*   DESIGN OF IRREDUNDANT MOS NETWORK        *
*                                            *
**********************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

**********************************************
```

NUMBER OF EXTERNAL VARIABLES = 3
NUMBER OF OUTPUT FUNCTIONS = 2

GIVEN FUNCTIONS ARE REALIZED AT LAST 2 CELLS

FUNCTION 1
10110011

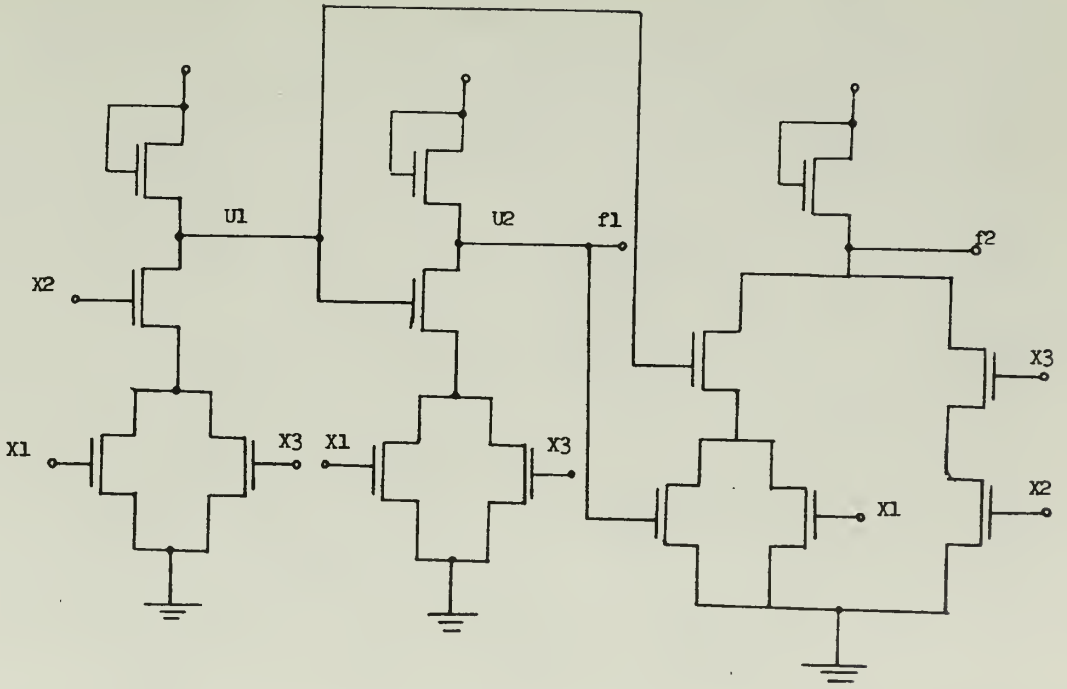
FUNCTION 2
01000010

NETWORK CONFIGURATION

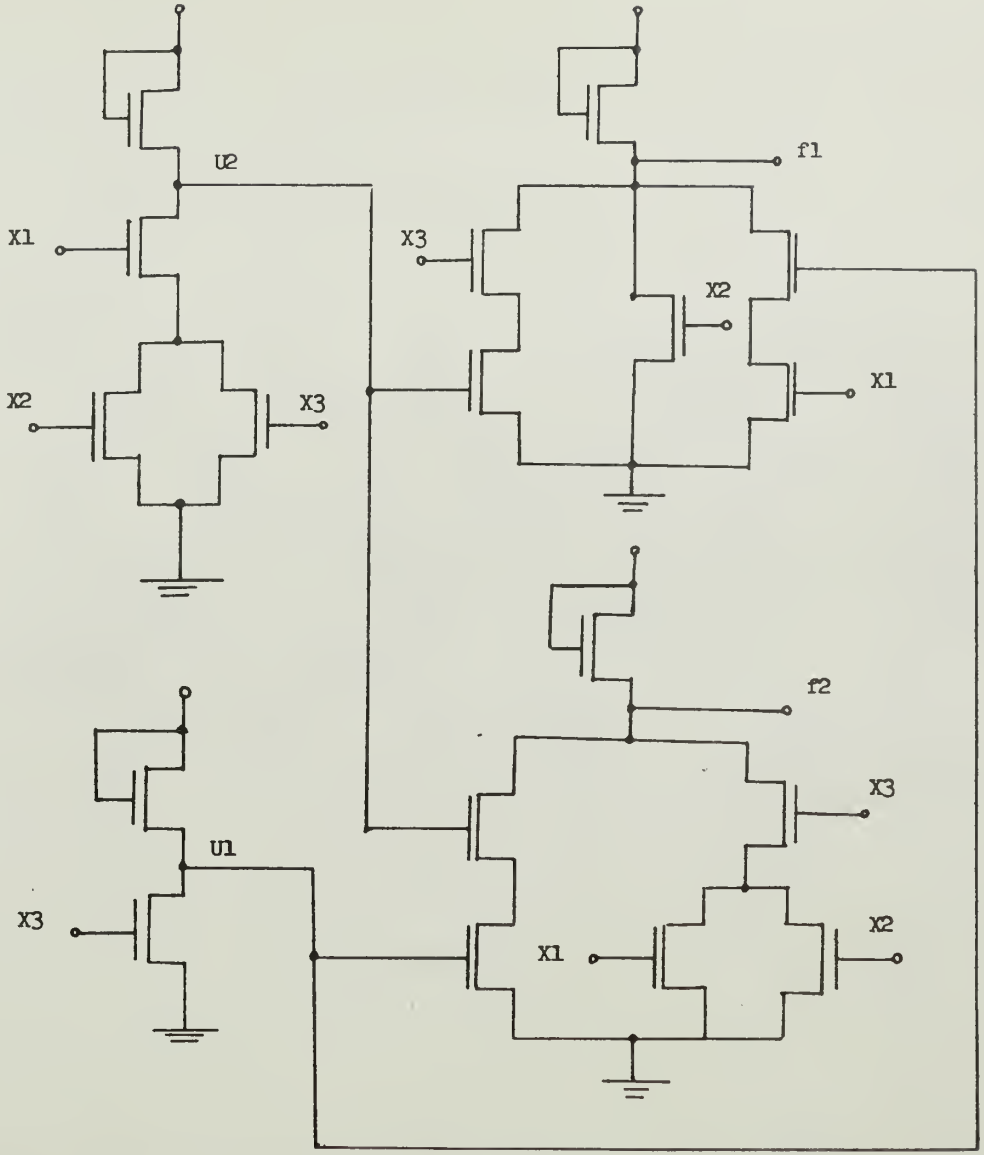
```
MOS CELL 1   0--|--X2--X3--|--0
                |--X1--X2--|

MOS CELL 2   0--|--X3--U1--|--0
                |--X1--U1--|

                |--U1--U2--|
MOS CELL 3   0--|--X2--X3--|--0
                |--X1--U1--|
```

NUMBER OF MOS CELLS = 3
NUMBER OF MOS FETS = 14
(WITHOUT FACTORING)

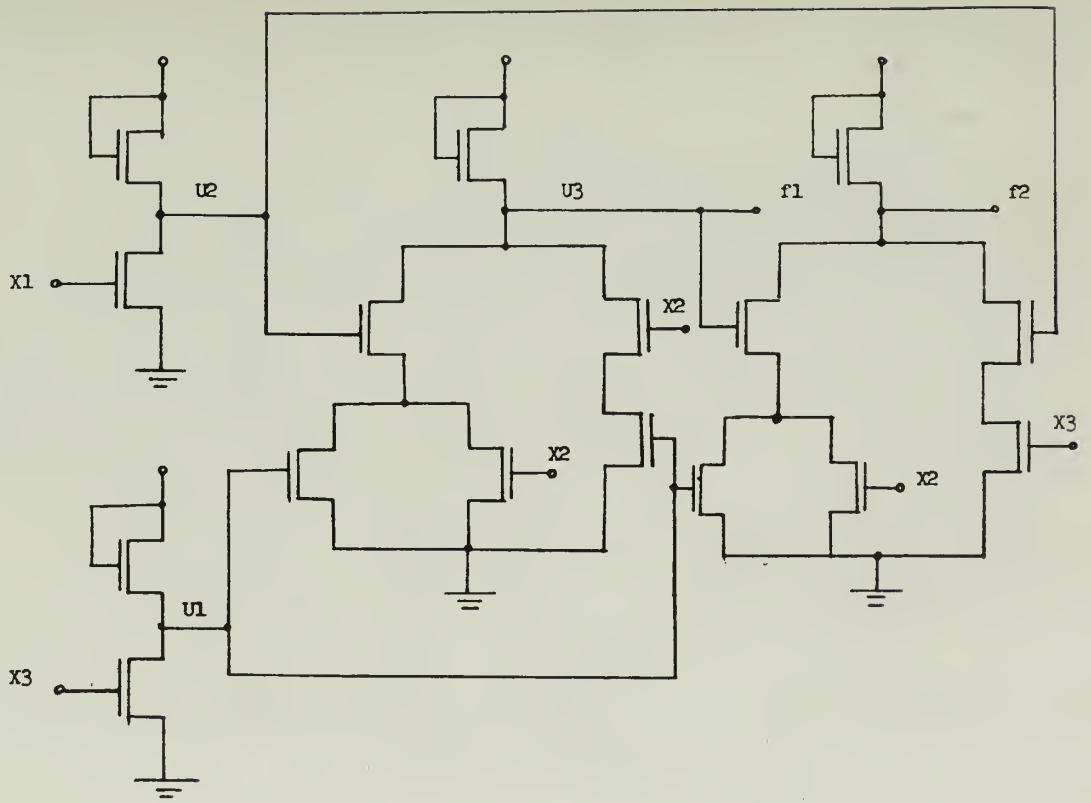
ELAPSED TIME = 0.09 SEC



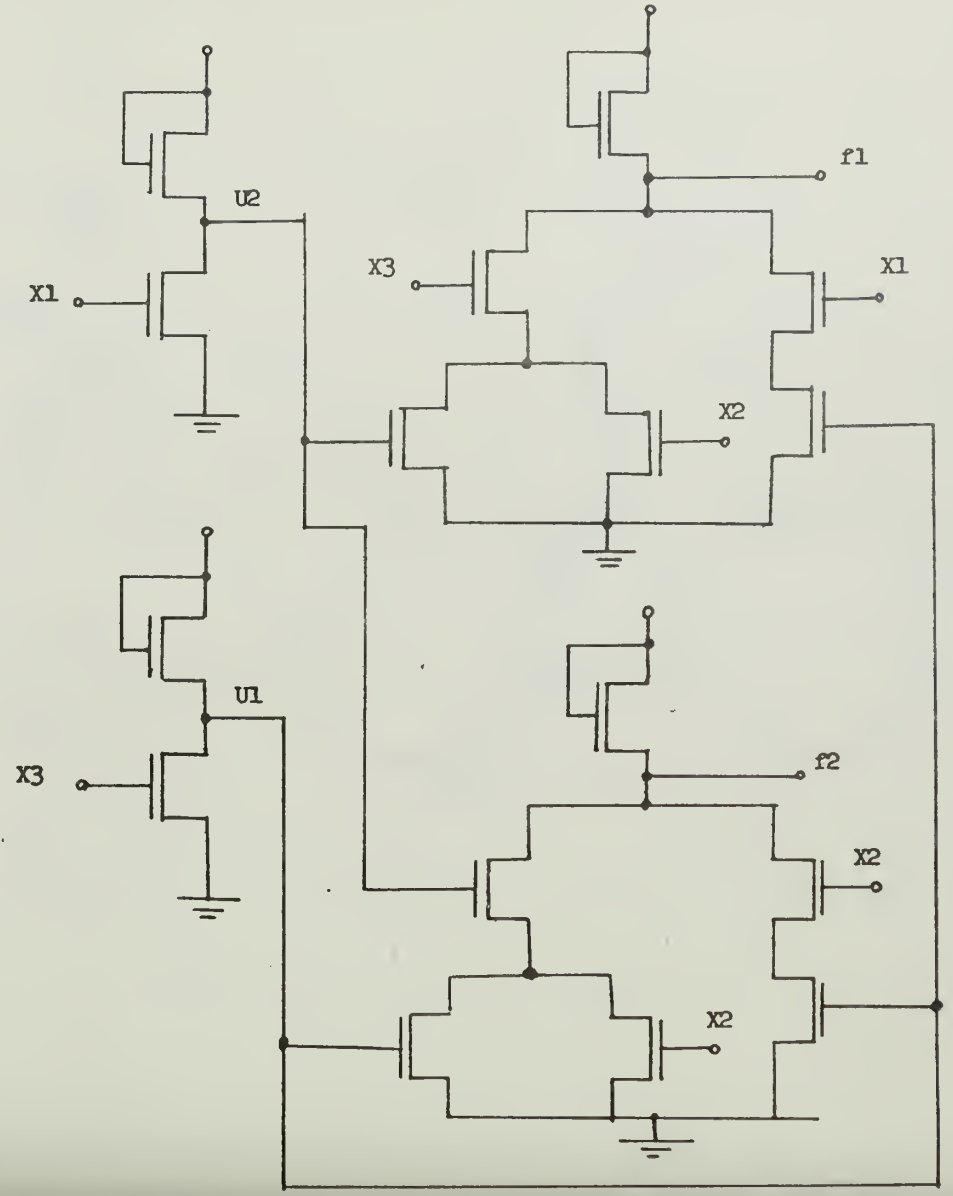
```
**************************************
*                                    *
*   DESIGN OF IRREDUNDANT MOS NETWORK *
*                                    *
**************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL
**************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST LEVEL

FUNCTION  1
10000100

FUNCTION  2
01000010

NETWORK CONFIGURATION

MOS CELL 1  0-----X3-----0

MOS CELL 2  0--|--X1--X3--|--0
               |--X1--X2--|

               |--X3--U2--|
MOS CELL 3  0--|--X1--U1--|--0
               |--X2------|

               |--U1--U2--|
MOS CELL 4  0--|--X2--X3--|--0
               |--X1--X3--|

NUMBER OF MOS CELLS =  4
NUMBER OF MOS FETS  = 16
(WITHOUT FACTORING)

ELAPSED TIME =   0.17 SEC
```



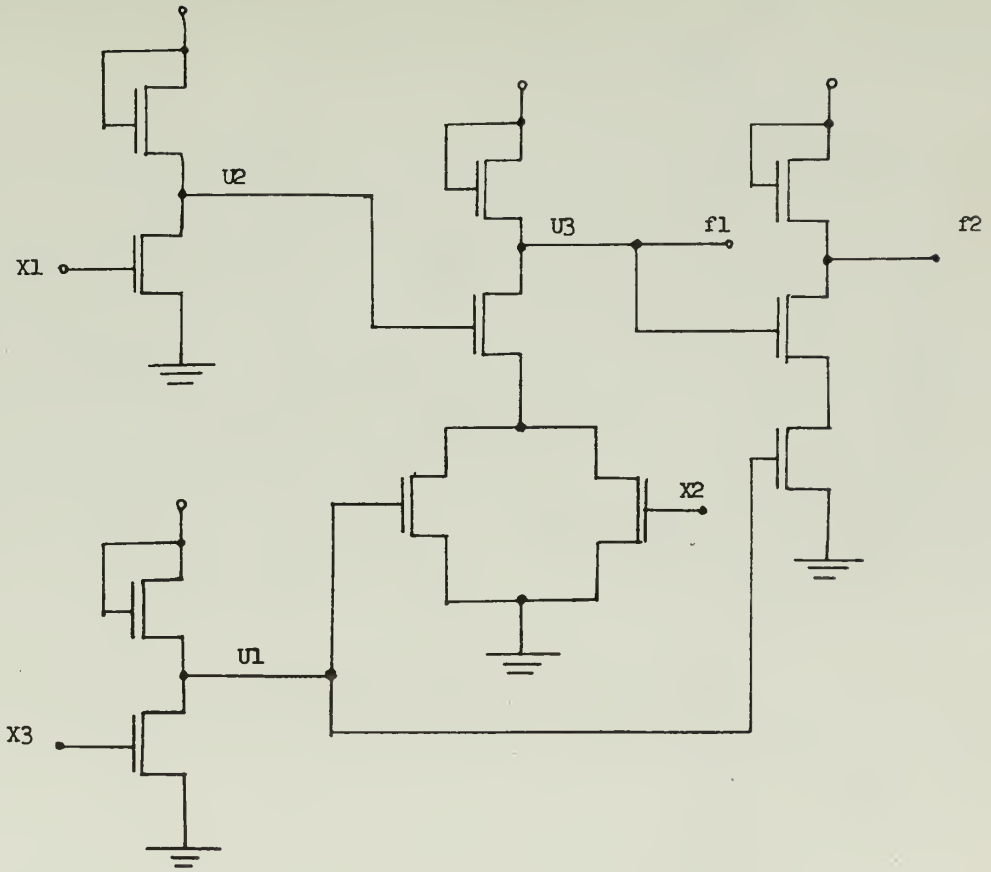
```
**********************************************
*   DESIGN OF IRREDUNDANT MOS NETWORK        *
**********************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL
**********************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST  2 CELLS

FUNCTION  1
01*01*01

FUNCTION  2
*01*01*0

NETWORK CONFIGURATION

MOS CELL 1   0-----X3-----0

MOS CELL 2   0-----X1-----0

                    |--U1--U2--|
MOS CELL 3   0--    |--X2--U2--|  --0
                    |--X2--U1--|

                    |--U1--U3--|
MOS CELL 4   0--    |--X3--U2--|  --0
                    |--X2--U3--|

NUMBER OF MOS CELLS =  4
NUMBER OF MOS SETS  = 14
(WITHOUT FACTORING)

ELAPSED TIME =    0.13 SEC
```



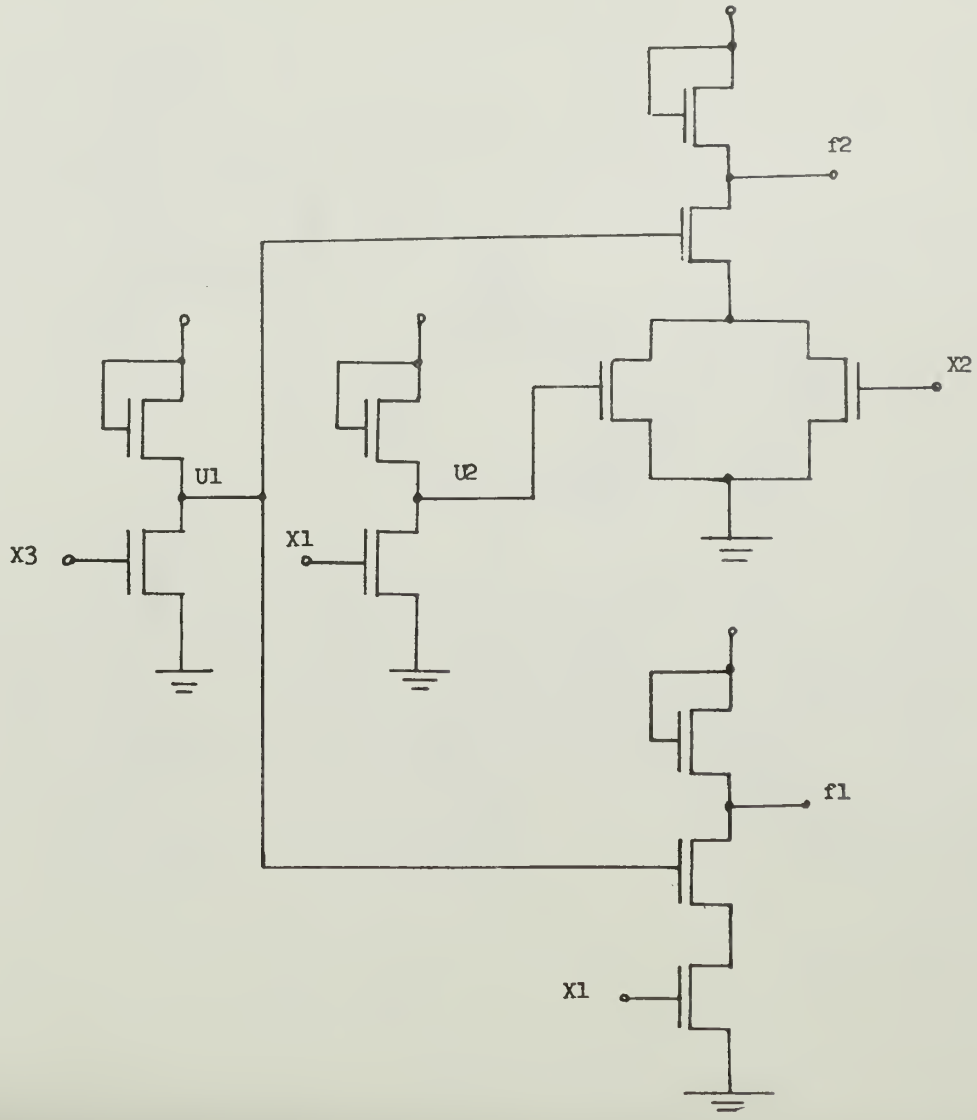
```
*****************************************
*                                       *
*   DESIGN OF IRREDUNDANT MOS NETWORK   *
*                                       *
*****************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

*****************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST LEVEL

FUNCTION  1
*01*01*0

FUNCTION  2
01*01*01

NETWORK CONFIGURATION

MOS CELL 1  0-----X3-----0

MOS CELL 2  0-----X1-----0

                  |--X3--U2--|
MOS CELL 3  0--   |--X2--X3--|   --0
                  |--X1--U1--|

                  |--U1--U2--|
MOS CELL 4  0--   |--X2--U2--|   --0
                  |--X2--U1--|

NUMBER OF MOS CELLS =  4
NUMBER OF MOS FETS  = 14
(WITHOUT FACTORING)

ELAPSED TIME =   0.13 SEC
```



```
**************************************
*                                    *
*   DESIGN OF IRREDUNDANT MOS NETWORK *
*                                    *
**************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

**************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST  2 CELLS

FUNCTION  1
010*1*01

FUNCTION  2
1**101**

NETWORK CONFIGURATION

MOS CELL 1   0-----X3-----0

MOS CELL 2   0-----X1-----0

                    |--U1--U2--|
MOS CELL 3   0--    |          |--0
                    |--X2--U1--|

MOS CELL 4   0-----U1--U3-----0

NUMBER OF MOS CELLS =  4
NUMBER OF MOS FETS  =  8
(WITHOUT FACTORING)

ELAPSED TIME =   0.11 SEC
```



```
**************************************
*                                    *
*   DESIGN OF IRREDUNDANT MOS NETWORK *
*                                    *
**************************************

   X = EXTERNAL VARIABLE
   U = OUTPUT OF MOS CELL

**************************************

NUMBER OF EXTERNAL VARIABLES =  3
NUMBER OF OUTPUT FUNCTIONS   =  2

GIVEN FUNCTIONS ARE REALIZED AT LAST LEVEL

FUNCTION  1
1**101**

FUNCTION  2
010*1*01

NETWORK CONFIGURATION

MOS CELL 1   0-----X3-----0

MOS CELL 2   0-----X1-----0

MOS CELL 3   0-----X1--U1-----0

MOS CELL 4   0--|--U1--U2--|--0
                |--X2--U1--|

NUMBER OF MOS CELLS =  4
NUMBER OF MOS FETS  =  8
(WITHOUT FACTORING)

ELAPSED TIME =   0.14 SEC
```



APPENDIX B

Program Listing

```
C   THIS PROGRAM IS THE IMPLEMENTATION OF LAI'S ALGORITHM, THAT IS,
C   ALGORITHM DIMN AND MODIFIED ALGORITHM DIMN ( DESIGN OF IRREDUNDANT
C   MOS NETWORK ).
C   THIS PROGRAM IS APPLICABLE TO THE NETWORKS WITH MULTIPLE
C   INCOMPLETELY SPECIFIED OUTPUTS.
C   THE RELATION BETWEEN THE MAXIMUM NUMBER OF EXTERNAL VARIABLES AND
C   THE MAXIMUM NUMBER OF OUTPUTS WHICH THIS PROGRAM CAN HANDLE IS
C   SHOWN BELOW.
C
C   FOR SINGLE-OUTPUT FUNCTIONS, THE MAXIMUM NUMBER OF EXTERNAL
C   VARIABLES WHICH THIS PROGRAM CAN HANDLE IS 8 OR THE RESTRICTION
C   ON PROBLEM SIZE HAS TO SATISFY THE FOLLOWING RELATIONSHIPS ( L = 12
C   FOR THIS PROGRAM ).
C
C        N + 1 + LOG2( N/2 + 1 ) NOT GREATER THAN L  IF N IS EVEN
C     OR
C        N + 1 + LOG2(( N + 1 )/2 + 1 ) NOT GREATER THAN L IF N IS ODD
C   WHERE LOG2( X ) TAKES THE SMALLEST INTEGER NOT SMALLER THAN
C   LOG2( X ).
C
C   FOR MULTIPLE-OUTPUT FUNCTIONS WITH ALL OUTPUT FUNCTIONS REALIZED
C   AT LAST M LEVELS, THE RESTRICTION ON PROBLEM SIZE HAS TO SATISFY
C   THE FOLLOWING RELATIONSHIP.
C
C        N + M + LOG2( N + 1 ) NOT GREATER THAN L
C   OR THE TABLE SHOWN BELOW.
C
C              MAX. EXTERNAL VARIABLES       MAX. OUTPUTS
C                        7                        2
C                        6                        3
C                        5                        4
C                        4                        5
C                        3                        7
C                        2                        8
C
C   WHERE LOG2( X ) TAKES THE SMALLEST INTEGER NOT SMALLER THAN
C   LOG2( X ).
C
C   FOR MULTIPLE-OUTPUT FUNCTIONS WITH ALL OUTPUT FUNCTIONS REALIZED
C   AT LAST LEVEL, THE RESTRICTION ON PROBLEM SIZE HAS TO SATISFY THE
C   FOLLOWING RELATIONSHIP.
C
C        N + 1 + LOG2( N + 1 ) NOT GREATER THAN L
C
C   WHERE LOG2( X ) TAKES THE SMALLEST INTEGER NOT SMALLER THAN
C   LOG2( X ).
C
C   THIS PROGRAM IS CONSTRUCTED WITH THE FOLLOWING MAIN SUBROUTINES AND
C   THE OTHER SMALL SUBROUTINES.
C
C     SUBROUTINE NAME    OPERATION
C
C          NCUBE         CONSTRUCT N-CUBE ACORDING TO THE NUMBER OF
C                        EXTERNAL VARIABLES.
C          INPUT         READ DATA INTO N-CUBE ( LABEL, DCARE ).
C          CMNL          IMPLEMENT CONDITIONAL MINIMUM LABELING.
C          CMXL          IMPLEMENT CONDITIONAL MAXIMUM LABELING.
C          MPF           OBTAIN MAXIMUM PERMISSIBLE FUNCTION FROM
C                        MINIMUM LABEL AND MAXIMUM LABEL.
```

```
C          IMC              OBTAIN IRREDUNDANT MOS CELL CONFIGURATION
C                           FROM MAXIMUM PERMISSIBLE FUNCTION.
      IMPLICIT INTEGER( A-Z )
      INTEGER*2  LABEL  ,DCARE  ,MNL    ,MXL   ,CHAIN  ,FUNC  ,LINK
     1         ,MINV
      COMMON     II     ,N       ,M      ,RF              ,FL
     1  ,LABEL(1024)    ,DCARE(1024)    ,MNL(1024)       ,MXL(1024)
     2  ,CHAIN(1024)    ,STARTL(11)     ,ENDL(11)        ,FUNC(4096)
     3  ,LINK(4096)     ,MINV(4096)     ,STARTF(15)      ,ENDF(15)
      LOGICAL  CS
      REAL UTIME,AA,BB
C     II                POINTER WHICH INDICATES THE CUPRENT STEP.
C     N                 NUMBER OF EXTERNAL VARIABLES.
C     M                 NUMBER OF OUTPUT FUNCTIONS.
C     FL                INDICTOR WHICH INDICATES THE CASES OF MULTIPLE-
C                       OUTPUT FUNCTIONS.
C     RF                MINIMUM NUMBER OF MOS CELLS
C     CS                IS SET IN SUBROUTINE-INPUT IF THE GIVEN OUTPUT
C                       FUNCTIONS ARE COMPLETELY SPECIFIED.
C     UTIME             STORES THE TIME ELAPSED FOR CONSTRUCTING NETWORK.
C     TFET              STORES TOTAL NUMBER OF DRIVER MOS FETS IN NETWORK.
C     LABEL(1024)       STORES LABEL AND FUNCTION VALUE WHICH IS ASSIGNED
C                       TO EACH VERTEX IN N-CUBE.
C     DCARE(1024)       STORES THE DONTCARE BIT OF OUTPUT FUNCTIONS.
C     MNL  (1024)       STORES THE LABEL ASSIGNED TO EACH VERTEX BY
C                       CONDITIONAL MINIMUM LABELING.
C     MXL  (1024)       STORES THE LABEL ASSIGNED TO EACH VERTEX BY
C                       CONDITIONAL MAXIMUM LABELING.
C     CHAIN(1024)       STORES THE LINK TO THE NEXT VERTEX WITH THE SAME
C                       WEIGHT IN N-CUBE.
C     STARTL(11)        POINTER TO THE FIRST VERTEX WITH THE SAME WEIGHT
C                       IN N-CUBE.
C     ENDL  (11)        POINTER TO THE LAST VERTEX WITH THE SAME WEIGHT IN
C                       N-CUBE.
C     FUNC(4096)        STORES MAXIMUM PERMISSIBLE FUNCTION ASSIGNED TO EACH
C                       VERTEX IN LARGE CUBE FOR OBTAINING IRREDUNDANT
C                       MOS CELL CONFIGURATION.
C     LINK(4096)        STORES THE LINK TO THE NEXT VERTEX WITH THE SAME
C                       WEIPHT IN LARGE CUBE.
C     MINV(4096)        STORES THE MINIMUM VECTOR OBTAINED IN THE PPOCESS
C                       OF IMC.
      CONTINUE
C     STARTF(15)        POINTER TO THE FIRST VERTEX WITH THE SAME WEIGHT
C                       IN LARGE CUBE.
C     ENDF  (15)        POINTER TO THE LAST VERTEX WITH THE SAME WEIGHT
C                       IN LARGE CUBE.
C**********************
C                      *
C     MAIN PROCEDURE   *
C                      *
C**********************
C
C     READ PARAMETERS N, M, AND FL.
C
   10 READ( 5,11,END=130 )N,M,FL
   11 FORMAT( I2,2X,I2,2X,I1 )
C
C     PRINT HEADING
C
      PRINT 12
   12 FORMAT( '1','*****************************************' )
```

```
      PRINT 13
   13 FORMAT( ' ','*                                          *' )
      PRINT 14
   14 FORMAT( ' ','*   DESIGN OF IRREDUNDANT MOS NETWORK    *' )
      PRINT 15
   15 FORMAT( ' ','*                                          *' )
      PRINT 16
   16 FORMAT( ' ','**********************************************' )
      PRINT 17
   17 FORMAT( '0','    X = EXTERNAL VARIABLE    ' )
      PRINT 18
   18 FORMAT( ' ','    U = OUTPUT OF MOS CELL    ' )
      PRINT 19
   19 FORMAT( '0','**********************************************' )
C
CC    CHECK PARAMETER VALUE
C
      IF( M .EQ. 1 ) GO TO 22
      AA = N + 1
      AA = ALOG10( AA )
      BB = ALOG10( 2.0 )
      A = AA/BB
      IF( FL .EQ. 1 ) GO TO 21
      A = N + M + A
   20 IF( A .GT. 12 ) GO TO 25
      GO TO 30
   21 A = N + 1 + A
      GO TO 20
   22 IF( N/2*2 .EQ. N ) GO TO 24
      AA = N/2 + 1
   23 AA = ALOG10( AA )
      BB = ALOG10( 2.0 )
      A = AA/BB
      A = N + 1 + A
      IF( A .GT. 12 ) GO TO 25
      GO TO 30
   24 AA = ( N + 1 )/2 + 1
      GO TO 23
C
CC    ERROR IN PARAMETER VALUE
C
   25 PRINT 26, N, M, FL
   26 FORMAT( '0','INPUT ERROR IN PARAMETER CARD',3X,'N =',I3,'M =',I3,
     1   'FL =',I3 )
      GO TO 130
C
CC    PRINT PARAMETER VALUE
C
   30 PRINT 31, N
   31 FORMAT( '-','NUMBER OF EXTERNAL VARIABLES =',I3 )
      PRINT 32, M
   32 FORMAT( ' ','NUMBER OF OUTPUT FUNCTIONS   =',I3 )
      IF( M .EQ. 1 ) GO TO 36
      IF( FL .EQ. 1 ) GO TO 34
      PRINT 33, M
   33 FORMAT( '-','GIVEN FUNCTIONS ARE REALIZED AT LAST ',I2,' CELLS' )
      GO TO 36
   34 PRINT 35
   35 FORMAT( '-','GIVEN FUNCTIONS ARE REALIZED AT LAST LEVEL ')
C
C     INITIALIZE LABEL( 2**N ) AND DCARE( 2**N )
```

```
C
   36 NVTEXL = 2**N
      DO 50 I = 1,NVTEXL
         LABEL( I ) = 0
         DCARE( I ) = 0
   50 CONTINUE
      CALL INPUT( CS,&130 )
C
CCC   SET TIMER
C
      CALL STEPZ( TIME )
      CALL NCUBE
      II = 1
      RF = 16
      TFET = 0
      PRINT 51
   51 FORMAT( '-','NETWORK CONFIGURATION' )
   60 CALL CMNL
      IF( FL .EQ. 1 ) GO TO 65
      IF( RF .EQ. 1 ) GO TO 84
      GO TO 66
   65 IF( RF .EQ. M ) GO TO 90
   66 CALL CMXL
C
CCCC  IF MNL( I ) = MXL( I ) FOR EVERY I IN N-CUBE, CMNL, CMXL,MPF
CCCC  CAN BE SKIPPED.
C
      IF( FL .EQ. 1 ) GO TO 68
      DO 67 I = 1,NVTEXL
         IF( MNL( I ) .NE. MXL( I ) ) GO TO 80
   67 CONTINUE
      GO TO 120
   68 DO 70 I = 1,NVTEXL
         MNLY = MNL( I )/2**M
         MXLY = MXL( I )/2**M
         IF ( MNLY .NE. MXLY ) GO TO 80
   70 CONTINUE
      GO TO 120
   80 CALL MPF
      CALL IMC( TFET,&10 )
      II = II + 1
      IF( FL .EQ. 1 ) GO TO 85
      IF( CS ) GO TO 85
      IF( II .NE. RF ) GO TO 60
   84 CALL ASFUN1
      CALL IMC( TFET,&10 )
      GO TO 125
C
CCCC  DECIDE THE MOS CELL CONFIGURATIONS IN COMPLETELY SPECIFIED
CCCC  FUNCTION PART
C
   85 IF( RF - II + 1 .GT. M ) GO TO 60
   90 CALL ASFUN1
      CALL IMC( TFET,&10 )
      II = II + 1
      IF ( II .LE. RF ) GO TO 90
      GO TO 125
C
CCC   MNL( I ) = MXL( I ) OCCURED FOR EVERY I IN N-CUBE
C
  120 CALL ASFUN2
```

```
      CALL IMC( TFET,810 )
      II = II + 1
      IF( FL .EQ. 1 ) GO TO 124
      IF( II .GT. RF ) GO TO 125
      GO TO 120
  124 IF( RF - II + 1 .GT. M ) GO TO 120
      GO TO 90
C
C   PRINT STATISTICS
C
  125 PRINT 126,RF
  126 FORMAT( '-','NUMBER OF MOS CELLS =',I3 )
      PRINT 127,TFET
  127 FORMAT( ' ','NUMBER OF MOS FETS  =',I3 )
      PRINT 128
  128 FORMAT( ' ','(WITHOUT FACTORING)' )
      CALL STEP2( ITIME )
      UTIME = ( TIME - ITIME )/100.
      PRINT 129,UTIME
  129 FORMAT( '0','ELAPSED TIME =',F7.2,' SEC ' )
      GO TO 10
  130 STOP
      END
C************************
C                       *
C   SUBROUTINE ASFUN1   *
C                       *
C************************
      SUBROUTINE ASFUN1
C   THIS SUBROUTINE ASSIGNS THE LABEL OR FUNCTION VALUE FROM N-CUBE
C   TO LARGE-CUBE( FUNC ).
      IMPLICIT INTEGER( A-Z )
      INTEGER*2  LABEL  ,DCARE  ,MNL    ,MXL   ,CHAIN  ,FUNC  ,LINK
     1          ,MINV
      COMMON     II     ,N      ,M      ,RF              ,FL
     1  ,LABEL(1024)    ,DCARE(1024)    ,MNL(1024)       ,MXL(1024)
     2  ,CHAIN(1024)    ,STARTL(11)     ,ENDL(11)        ,FUNC(4096)
     3  ,LINK(4096)     ,MINV(4096)     ,STARTF(15)      ,ENDF(15)
      MXVTXL = 2**N - 1
      RSHIFT = 2**( RF - II )
      IF( FL .EQ. 1 ) GO TO 5
      NVTEXP = 2**( N + II - 1 )
      LSHIFT = 2**( II - 1 )
      SCALE = 2*RSHIFT
      GO TO 8
    5 NVTEXP = 2**( N + RF - M )
      LSHIFT = 2**( RF - M )
      SCALE = 2**M
    8 DO 10 I = 1,NVTEXP
   10 FUNC( I ) = 0
      I = 0
   20 LABELX = LABEL( I + 1 ) / RSHIFT
      DCARX = DCARE( I + 1 ) / RSHIFT
      J = I*LSHIFT + LABEL( I + 1 )/SCALE
      IF( DCARX/2*2 .NE. DCARX ) GO TO 40
      IF( LABELX/2*2 .EQ. LABELX ) GO TO 30
      FUNC( J + 1 ) = 2
      GO TO 40
   30 FUNC( J + 1 ) = 1
   40 I = I + 1
      IF( I .LE. MXVTXL ) GO TO 20
```

```
      RETURN
      END
C*************************
C                        *
C   SUBROUTINE ASFUN2    *
C                        *
C*************************
      SUBROUTINE ASFUN2
C   THIS SUBROUTINE ASSIGNS THE LABEL OR FUNCTION VALUE FROM
C   MNL( N-CUBE ) TO LARGE-CUBE( FUNC ).
      IMPLICIT INTEGER( A-Z )
      INTEGER*2  LABEL  ,DCARE  ,MNL    ,MXL   ,CHAIN  ,FUNC  ,LINK
     1          ,MINV
      COMMON     II     ,N       ,M      ,RF              ,FL
     1  ,LABEL(1024)    ,DCARE(1024)     ,MNL(1024)       ,MXL(1024)
     2  ,CHAIN(1024)    ,STARTL(11)      ,ENDL(11)        ,FUNC(4096)
     3  ,LINK(4096)     ,MINV(4096)      ,STARTF(15)      ,ENDF(15)
      MXVTXL = 2**N - 1
      NVTEXF = 2**( N + II - 1 )
      RSHIFT = 2**( RF - II )
      LSHIFT = 2**( II - 1 )
      DO 10 I = 1,NVTEXF
   10 FUNC( I ) = 0
      I = 0
   20 MNLX = MNL( I + 1 ) / RSHIFT
      J = I*LSHIFT + MNLX/2
      IF( MNLX/2*2 .EQ. MNLX ) GO TO 30
      FUNC( J + 1 ) = 2
      GO TO 40
   30 FUNC( J + 1 ) = 1
   40 I = I + 1
      IF( I .LE. MXVTXL ) GO TO 20
      RETURN
      END
C*************************
C                        *
C   SUBROUTINE ASIGN1    *
C                        *
C*************************
      SUBROUTINE ASIGN1( USPFY,M,LBY,LBY )
C   THIS SUBROUTINE ASSIGNS ONE TO THE EVERY UNSPECIFIED BIT OF LABELS
C   IN N-CUBE.
      IMPLICIT INTEGER( A-Z )
      SHIFT = 1
      IF( USPFY .GT. M ) GO TO 30
      DO 20 I = 1,USPFY
         IF( LBY/2*2 .EQ. LBY ) GO TO 10
         LBX = LBX + SHIFT
   10    SHIFT = 2 * SHIFT
         LBY = LBY / 2
   20 CONTINUE
      GO TO 70
   30 DO 50 I = 1,M
         IF( LBY/2*2 .EQ. LBY ) GO TO 40
         LBX = LBY + SHIFT
   40    SHIFT = 2 * SHIFT
         LBY = LBY /2
   50 CONTINUE
      L = M + 1
      DO 60 I = L,USPFY
         LBX = LBX + SHIFT
```

```
          SHIFT = 2 * SHIFT
    60 CONTINUE
    70 RETURN
       END
C************************
C                       *
C    SUBROUTINE INCMNT  *
C                       *
C************************
       SUBROUTINE INCMNT( INB,NDCARE,LBX,M,BWEIT )
C    THIS SUBROUTINE INCREMENTS THE LABEL ASSIGNED TO THE VERTEX POINTED
C    BY POINTER-PTRB.
       IMPLICIT INTEGER( A-Z )
       INTEGER BWEIT( 10 )
C    INB IS THE COUNTER WHICH HAS SPECIFIED WEIGHT FOR EACH BIT.
       INB = INB + 1
       INBX = INB
       J = 1
       SHIFT = 2 ** ( M - NDCARE )
    10 IF( INBX .EQ. 0 ) GO TO 40
       IF( INBX/2*2 .EQ. INBX ) GO TO 30
       IF( J .LE. NDCARE ) GO TO 20
       LBX = LBX + SHIFT
       GO TO 30
    20 LBX = LBX + BWEIT( J )
    30 J = J + 1
       SHIFT = 2 * SHIFT
       INBX = INBX / 2
       GO TO 10
    40 RETURN
       END
C************************
C                       *
C    SUBROUTINE DCRMNT  *
C                       *
C************************
       SUBROUTINE DCRMNT( INB,NDCARE,LBXX,M,BWEIT )
C    THIS SUBROUTINE DECREMENT THE LABEL ASSIGNED TO THE VERTEX POINTED
C    BY POINTER-PTRB
       IMPLICIT INTEGER( A-Z )
       INTEGER BWEIT( 10 )
       INB = INB + 1
       INBX = INB
       J = 1
       SHIFT = 2 ** ( M - NDCARE )
    10 IF( INBX .EQ. 0 ) GO TO 40
       IF( INBX/2*2 .EQ. INBX ) GO TO 30
       IF( J .LE. NDCARE ) GO TO 20
       LBXX = LBXX - SHIFT
       GO TO 30
    20 LBXX = LBXX - BWEIT( J )
    30 J = J + 1
       SHIFT = 2 * SHIFT
       INBX = INBX / 2
       GO TO 10
    40 RETURN
       END
C************************
C                       *
C    SUBROUTINE DSCAN   *
C                       *
```

```
C************************
      SUBROUTINE DSCAN( LBY,USPFY,NDCARE,BWEIT )
C    THIS SUBROUTINE SCAN THE DCARE WHICH IS ASSIGNED TO EACH VERTEX
C    AND OBTAIN THE NUMBER OF DONTCARE BITS AND THE WEIGHT ASSIGNED
C    TO EACH DONTCARE BIT.
      IMPLICIT INTEGER( A-Z )
      INTEGER BWEIT(10)
C    BWEIT( 10 )        STORES THE WEIGHT ASSIGNED TO EACH DONTCARE BIT.
C    NDCARE             STORES THE NUMBER OF DONTCARE BITS IN ONE VERTEX.
      NDCARE = 0
      DO 5 I = 1,10
    5 BWEIT( I ) = 0
      I = 1
      SHIFT = 1
      DO 20 I = 1,USPFY
         IF( LBY .EQ. 0 ) GO TO 30
         IF( LBY/2*2 .EQ. LBY ) GO TO 10
         NDCARE = NDCARE + 1
         BWEIT( NDCARE ) = SHIFT
   10    SHIFT = 2 * SHIFT
         LBY = LBY / 2
   20 CONTINUE
   30 RETURN
      END
C************************
C                        *
C    SUBROUTINE CMNL     *
C                        *
C************************
      SUBROUTINE CMNL
C    THIS SUBROUTINE IMPLEMENTS CONDITIONAL MINIMUM LABELING.
C    THIS SUBROUTINE CALLS SUBROUTINE-DSCAN AND SUBROUTINE-INCMNT.
      IMPLICIT INTEGER( A-Z )
      INTEGER*2  LABEL  ,DCARE  ,MNL    ,MXL   ,CHAIN  ,FUNC  ,LINK
     1         ,MINV
      COMMON     II     ,N      ,M      ,RF              ,FL
     1  ,LABEL(1024)    ,DCARE(1024)    ,MNL(1024)       ,MXL(1024)
     2  ,CHAIN(1024)    ,STARTL(11)     ,ENDL(11)        ,FUNC(4096)
     3  ,LINK(4096)     ,MINV(4096)     ,STARTF(15)      ,ENDF(15)
      INTEGER BWEIT (10)
C    PTRB POINTS TO THE VERTEX-B FOR WHICH WE ARE SEEKING FOR THE
C    MINIMUM POSSIBLE LABEL.
C    PTRA POINTS TO THE VERTEX-A WHICH HAS BIGGER WEIGHT BY ONE THAN
C    VERTEX-B AND CONNECTED TO VERTEX-B BY THE EDGE IN N-CUBE.
C    RF( MINIMUM NUMBER OF MCS CELLS ) IS OBTAINED IN THE FIRST EXECUTION
C    OF THIS SUBROUTINE.
      F = M
      MNL( STARTL( N + 1 ) ) = LABEL( STARTL( N + 1 ) )
      USPFY = RF - II + 1
      W = N
   10 PTRB = STARTL( W )
   20 IBX = LABEL( PTRB )
      LBY = DCARE( PTRB )
      SHIFT = 1
      INB = 0
      CALL DSCAN( LBY,USPFY,NDCARE,BWEIT )
      DO 160  I=1, N
         PTRBY=(PTRB-1)/SHIFT
         IF(PTRBY/2*2.NE.PTRBX)  GO TO 150
         PTRA=PTRB+SHIFT
         LAX=MNL(PTRA)
```

```
      IF( FL .EQ. 1 ) GO TO 25
   22 IF( LBX .GE. LAX ) GO TO 150
      LBX = LABEL( PTRB )
      CALL INCMNT ( INB,NDCARE,LBX,F,BWEIT )
      GO TO 22
   25    LBZ=LBX/2**M
         LAZ=LAX/2**M
         L=0
         J=1
   30    LBT=LBX/2**L
         LAT=LAX/2**L
         IF( LBT/2*2 .EQ. LBT ) GO TO 40
         LB=2*LBZ+1
         GO TO 50
   40    LB=2*LBZ
   50    IF (LAT/2*2.EQ.LAT) GO TO 60
         LA=2*LAZ+1
         GO TO 70
   60    LA=2*LAZ
   70    IF ( LB .GE. LA ) GO TO 130
         IF (BWEIT(J).EQ.2**L) GO TO 90
   80    LB=LB+2
         LBX=LBX+2**M
         IF (LB.GE.LA) GO TO 140
         GO TO 80
   90    INC = 0
         LBX=LBX+BWEIT(J)
  100    LB=LB+1
         INC = INC + 1
         IF(LB.GE.LA) GO TO 120
         IF(INC/2*2 .EQ. INC ) GO TO 110
         LBX=LBX+2**M-BWEIT(J)
         GO TO 100
  110    LBX=LBX+BWEIT(J)
         GO TO 100
  120    J=J+1
         GO TO 140
  130    IF (BWEIT(J).NE.2**L) GO TO 140
         J=J+1
  140    L=L+1
         LBZ=LB/2
         IF (L.GE.M) GO TO 150
         GO TO 30
  150    SHIFT=2*SHIFT
  160 CONTINUE
      MNL(PTRB)=LBX
      IF (PTRB.EQ.ENDL(W)) GO TO 170
      PTRB=CHAIN (PTRB)
      GO TO 20
  170 W=W-1
      IF( W .GE. 1 ) GO TO 10
      IF( II .NE. 1 ) GO TO 200
      RP=0
      MNLX=MNL(1)
  180 IF( MNLX .EQ. 0 ) GO TO 200
      MNLX=MNLX/2
      RP=RP+1
      GO TO 180
  200 RETURN
      END
C**********************
```

```
C                        *
C    SUBROUTINE CMYL     *
C                        *
C************************
      SUBROUTINE CMYL
C     THIS SUBROUTINE IMPLEMENTS CONDITIONAL MAXIMUM LABELING.
C     THIS SUBROUTINE CALLS SUBROUTINE-DSCAN, SUBROUTINE-ASIGN1 AND
C     SUBROUTINE-DCRMNT.
      IMPLICIT INTEGER( A-Z )
      INTEGER*2  LABEL  ,DCARE  ,MNL    ,MXL   ,CHAIN  ,FUNC  ,LINK
     1          ,MINV
      COMMON     II     ,N       ,M      ,RF              ,FL
     1  ,LABEL(1024)    ,DCARE(1024)    ,MNL(1024)       ,MXL(1024)
     2  ,CHAIN(1024)    ,STARTL(11)     ,ENDL(11)        ,FUNC(4096)
     3  ,LINK(4096)     ,MINV(4096)     ,STARTF(15)      ,ENDF(15)
      INTEGER BWEIT(10)
C     PTRB POINTS TO THE VERTEX-B FOR WHICH WE ARE SEEKING FOR THE
C     MAXIMUM POSSIBLE LABEL.
C     PTRA POINTS TO THE VERTEX-A WHICH HAS SMALLER WEIGHT BY ONE THAN
C     VERTEX-B AND CONNECTED TO VERTEX-B BY THE EDGE IN N-CUBE.
      F = M
      LBX = LABEL( STARTL( 1 ) )
      LBY = DCARE( STARTL( 1 ) )
      USPFY = RF - II + 1
      CALL ASIGN1( USPFY,F,LBX,LBY )
      MXL( STARTL( 1 ) ) = LBX
      K = N + 1
      DO 170 W = 2,K
         PTRB = STARTL( W )
   20    LBY = DCARE( PTRB )
         CALL DSCAN( LBY,USPFY,NDCARE,BWEIT )
         LBX = LABEL( PTRB )
         LBY = DCARE( PTRB )
         CALL ASIGN1( USPFY,F,LBX,LBY )
         LBXX = LBY
         SHIFT = 1
      INB = 0
         DO 160 I=1, N
            PTRBY=(PTRB-1)/SHIFT
            IF (PTRBY/2*2.EQ.PTRBY) GO TO 150
            PTRA=PTRB-SHIFT
            LAX = MYL( PTRA )
      IF( FL .EQ. 1 ) GO TO 25
   22 IF( LBXX .LE. LAX ) GO TO 150
      LBXX = LBX
      CALL DCRMNT( INB,NDCARE,LBXX,F,BWEIT )
      GO TO 22
   25       LBZ=LBXX/2**M
            LAZ=LAX/2**M
            L=0
            J=1
   30       LBT=LBXX/2**L
            LAT=LAX/2**L
            IF (LBT/2*2.EQ.LBT) GO TO 40
            LB=2*LBZ+1
            GO TO 50
   40       LB=2*LBZ
   50       IF (LAT/2*2.EQ.LAT) GO TO 60
            LA=2*LAZ+1
            GO TO 70
   60       LA=2*LAZ
```

```
   70         IF (LB.LE.LA) GO TO 130
              IF (BWEIT (J).EQ.2**L) GO TO 90
   80         LB=LB-2
              LBXX=LBXX-2**M
              IF (LB.LE.LA) GO TO 140
              GO TO 80
   90         LBXX=LBXX-BWEIT (J)
              INC = 0
  100         LB=LB-1
              INC = INC + 1
              IF (LB.LE.LA) GO TO 120
              IF( INC/2*2 .EQ. INC ) GO TO 110
              LBXX=LBXX-2**M+BWEIT (J)
              GO TO 100
  110         LBXX=LBXX-BWEIT (J)
              GO TO 100
  120         J=J+1
              GO TO 140
  130         IF (BWEIT (J).NE.2**L) GO TO 140
              J=J+1
  140         L=L+1
              LBZ=LB/2
              IF (L.GE.M) GO TO 150
              GO TO 30
  150         SHIFT=2*SHIFT
  160       CONTINUE
            MXL (PTRB)=LBXX
            IF (PTRB.EQ.ENDL(W)) GO TO 170
            PTRB=CHAIN (PTRB)
            GO TO 20
  170 CONTINUE
      RETURN
      END
C*********************
C                    *
C   SUBROUTINE IMC   *
C                    *
C*********************
      SUBROUTINE IMC( TFET,* )
C     THIS SUBROUTINE OBTAINS IRREDUNDANT MOS CELL CONFIGURATION FROM
C     THE MAXIMUM PERMISSIBLE FUNCTION.
C     THIS SUBROUTINE IS MADE OF THE FOLLOWING SIX STEPS.
C       STEP-1    CONSTRUCTS LARGE-CUBE FUNC( 2**(N + II - 1) ).
C       STEP-2    ASSIGNS "0" OR "1" TO THE VERTEX-X WITH DONTCARE SUCH
C                 THAT THERE EXISTS NO VERTEX-Y SATISFYING Y > X AND
C                 P( Y )  = "1".
C       STEP-3    OBTAINS THE SET OF MINIMUM VECTORS.
C       STEP-4    OBTAINS THE SUBSET OF THE SET OF MINIMUM VECTORS WHICH
C                 COVERS EVERY VERTEX WITH ORIGINAL "0".
C       STEP-5    OBTAINS AN IRREDUNDANT SUBSET FROM THE SUBSET OBTAINED
C                 IN STEP-4.
C       STEP-6    STORES THE RESULT TO THE VERTEX IN N-CUBE ( LABEL ).
C                 THIS STEP CONTAINS ERROR CHECKING ROUTINE.
C                 IF FUNCTION VALUES FROM MOS CELL CONFIGURATION OBTAINED
C                 IN PREVIOUS STEPS ARE DIFFERENT FROM THE FUNCTION VALUES
C                 ALREADY STORED IN ARRAY-LABEL, ERROR MESSAGE AND
C                 THE CONTENTS OF ARRAY-LABEL IS PRINTED.
      IMPLICIT INTEGER( A-Z )
     *INTEGER*2  LABEL  ,DCARE  ,MNL    ,MXL   ,CHAIN  ,FUNC  ,LINK
     *          ,MINV
      INTEGER PRTARY( 20 )
```

```
      LOGICAL ERROR
      LOGICAL REPEAT
      COMMON      II      ,N          ,M          ,RF                ,FL
     1  ,LABEL(1024)      ,DCARE(1024)      ,MNL(1024)         ,MXL(1024)
     2  ,CHAIN(1024)      ,STARTL(11)       ,ENDL(11)          ,FUNC(4096)
     3  ,LINK(4096)       ,MINV(4096)       ,STARTF(15)        ,ENDF(15)
      INTEGER X( 10 ) / '--X1','--X2','--X3','--X4','--X5',
     1                  '--X6','--X7','--X8','--X9','-X10', /,
     2          U( 10 ) / '--U1','--U2','--U3','--U4','--U5',
     3                  '--U6','--U7','--U8','--U9','-U10' /
      INTEGER BLANK/ '    ' / , FORM1/ '   1' / , FORM2/ '0--1' / ,
     1          FORM3/ '--1 ' / , FORM4/ '  1 ' / , FORM5/ '--|-' / ,
     2          FORM6/ '-0  ' / , FORM7/ '-1--' / , FORM8/ '  |-' / ,
     3          FORM9/ '0---' /
C     PRTARY( 20 )      PRINT BUFFER FOR PRINTING OBTAINED MOS GATE
CC                      CONFIGURATION.
C     ERROR             IS SET IF ERROR OCCURS IN CONSTRUCTING NETWORK.
      RSHIFT = 2 ** ( RF - II )
      USPFY = RF - II + 1
      IF( FL .EQ. 1 ) GO TO 4
      GO TO 5
    4 IF ( USPFY .GT. M ) GO TO 5
      INPD = N + RF - M
      SCALE = 2**M
      LSHIFT = 2**( RF - M )
      GO TO 8
    5 INPD = N + II - 1
      SCALE = 2*RSHIFT
      LSHIFT = 2 ** ( II - 1 )
    8 NVTEXL = 2 ** N
      MXVTXL = NVTEXL - 1
      NVTEXF = 2 ** INPD
      MXVTXF = NVTEXF - 1
      ERROR = .FALSE.
C
CC    STEP-1
C
      L = INPD + 1
      DO 10 I = 1,L
   10 STARTF( I ) = 0
      I = 0
   20 Y = I
      W = 0
   30 IF( Y .EQ. 0 ) GO TO 50
      IF( Y/2*2 .EQ. Y ) GO TO 40
      W = W + 1
   40 Y = Y / 2
      GO TO 30
   50 IF( STARTF( W + 1 ).EQ.0 ) GO TO 60
      LINK( ENDF( W + 1 ) ) = I + 1
      ENDF( W + 1 ) = I + 1
      GO TO 70
   60 STARTF( W + 1 ) = I + 1
      ENDF( W + 1 ) = I + 1
   70 I = I + 1
      IF( I.LE.MXVTXF ) GO TO 20
C
CC    STEP-2
C
      IF( FUNC( STARTF( INPD + 1 ) ) .NE. 0 ) GO TO 110
      FUNC( STARTF( INPD + 1 ) ) = 3
```

```
      W = INPD
      PTRB = STARTF( W )
      IF( FUNC( PTRB ).NE.0 ) GO TO 160
      SHIFT = 1
      DO 150 I = 1,INPD
         PTRBY = ( PTRB - 1 ) / SHIFT
         IF( PTRBY/2*2 .NE. PTRBY ) GO TO 140
         PTRA = PTRB + SHIFT
         IF( FUNC( PTRA ).NE.2 ) GO TO 140
         FUNC( PTRB ) = 2
         GO TO 160
  140    SHIFT = 2 * SHIFT
  150 CONTINUE
      FUNC( PTRB ) = 3
  160 IF( PTRB.EQ.ENDF( W ) ) GO TO 170
      PTRB = LINK( PTRB )
      GO TO 130
  170 W = W - 1
      IF( W.GE.1 ) GO TO 120
C
C     STEP-3
C
      K = 1
      NMINV = 0
      W = INPD + 1
  210 PTRB = STARTF( W )
  220 IF( FUNC( PTRB ).EQ.2 ) GO TO 250
      SHIFT = 1
      DO 240 I = 1,INPD
         PTRBY = ( PTRB - 1 ) / SHIFT
         IF( PTRBY/2*2 .EQ. PTRBY ) GO TO 230
         PTRA = PTRB - SHIFT
         IF( FUNC( PTRA ).NE.2 ) GO TO 250
  230    SHIFT = 2 * SHIFT
  240 CONTINUE
      MINV( K ) = PTRB - 1
      NMINV = NMINV + 1
      K = K + 1
  250 IF( PTRB.EQ.ENDF( W ) ) GO TO 260
      PTRB = LINK( PTRB )
      GO TO 220
  260 W = W - 1
      IF( W.GE.1 ) GO TO 210
C
C     STEP-4
C
      DO 310 I = 1,NVTEXP
  310 LINK( I ) = 0
      I = 1
  320 REPEAT = .FALSE.
      DO 340 J = 1,NVTEXP
         IF( FUNC( J ).NE.1 ) GO TO 340
         JX = J - 1
         MINV = MINV( I )
         CALL COMPR( JX,MINV,&330 )
         LINK( J ) = LINK( J ) + 1
         GO TO 340
  330    IF( LINK( J ).NE.0 ) GO TO 340
         REPEAT = .TRUE.
  340 CONTINUE
      IF( REPEAT ) GO TO 350
```

```
      NSIRR = I
      GO TO 360
  350 I = I + 1
      GO TO 320
  360 CONTINUE
C
CC    STEP-5
C
      L = NSIRR - 1
      DO 450 I = 1,L
         DO 410 J = 1,NVTEXP
            IF( FUNC( J ).NE.1 ) GO TO 410
            JX = J - 1
            MINX = MINV( I )
            CALL COMPR( JX,MINX,&410 )
            LINK( J ) = LINK( J ) - 1
            IF( LINK( J ).EQ.0 ) GO TO 420
  410    CONTINUE
         GO TO 440
  420    IF( FUNC( J ).NE.1 ) GO TO 430
         JX = J - 1
         MINY = MINV( I )
         CALL COMPR( JX,MINY,&430 )
         LINK( J ) = LINK( J ) + 1
  430    J = J - 1
         IF( J.GE.1 ) GO TO 420
         GO TO 450
  440    MINV( I ) = -1
  450 CONTINUE
C
CC    PRINT MOS GATE CONFIGURATION
C
      NIRR = 0
      MXFET = 0
      DO 490 I = 1,NSIRR
         IF( MINV( I ) .LT. 0 ) GO TO 490
         NIRR = NIRR + 1
         MINX = MINV( I )
         NFET = 0
  460    IF( MINX .EQ. 0 ) GO TO 480
         IF( MINX/2*2 .EQ. MINX ) GO TO 470
         NFET = NFET + 1
  470    MINX = MINX / 2
         GO TO 460
  480    TFET = TFET + NFET
         IF( NFET .LE. MXFET ) GO TO 490
         MXFET = NFET
  490 CONTINUE
      PRINT 4100
 4100 FORMAT( '0' )
      I = 0
      L = 2 * NIRR - 1
      DO 4290 LINE = 1,L
         IF( LINE/2*2 .EQ. LINE ) GO TO 4240
C
CC    PRINT ODD LINE
C
 4101    I = I + 1
         IF( MINV( I ) .LT. 0 ) GO TO 4101
         IF( LINE .EQ. NIRR ) GO TO 4110
         PRTARY( 1 ) = FORM1
```

```
      GO TO 4120
4110  IF( NIRR .EQ. 1 ) GO TO 4111
      PRTARY( 1 ) = FORM2
      GO TO 4120
4111  PRTARY( 1 ) = FORM9
4120      K = 0
      IF( FL .EQ. 0 ) GO TO 4121
      IF( USPFY .GT. M ) GO TO 4121
      SHIFT = 2**( N + RF - M - 1 )
      GO TO 4122
4121  SHIFT = 2**( N + II - 2 )
4122  DO 4140 J = 1,N
          MINX = MINV( I ) / SHIFT
          IF( MINX/2*2 .EQ. MINX ) GO TO 4130
          K = K + 1
          PRTARY( K + 1 ) = X( J )
4130      SHIFT = SHIFT / 2
4140  CONTINUE
      IF( II .EQ. 1 ) GO TO 4170
      N1 = N + 1
      IF( FL .EQ. 0 ) GO TO 4143
      IF( USPFY .LE. M ) GO TO  4141
4143  N2 = N + II - 1
      GO TO 4142
4141  N2 = N + RF - M
4142  DO 4160 J = N1,N2
          MINX = MINV( I ) / SHIFT
          IF( MINX/2*2 .EQ. MINX ) GO TO 4150
          K = K + 1
          PRTARY( K + 1 ) = U( J - N )
4150      SHIFT = SHIFT / 2
4160  CONTINUE
4170  K = K + 1
      IF( K .GT. MYPET ) GO TO 4180
      PRTARY( K + 1 ) = FORM7
      GO TO 4170
4180  IF( LINE .EQ. NIRR ) GO TO 4210
      PRTARY( MYPET + 2 ) = FORM3
      PRTARY( MYPET + 3 ) = BLANK
4190  MX3 = MYPET + 3
      PRINT 4200,( PRTARY( JJ ),JJ = 1,MX3 )
4200  FORMAT( ' ',12X,20A4 )
      GO TO 4290
4210  IF( NIRR .EQ. 1 ) GO TO 4211
      PRTARY( MYPET + 2 ) = FORM5
      PRTARY( MYPET + 3 ) = FORM6
      GO TO 4220
4211  PRTARY( MYPET + 2 ) = FORM7
      PRTARY( MYPET + 3 ) = FORM6
4220  MX3 = MYPET + 3
      PRINT 4230,II,( PRTARY( JJ ),JJ = 1,MX3 )
4230  FORMAT( ' ','MOS CELL',I2,2X,20A4 )
      GO TO 4290
C
CC    PRINT EVEN LINE
C
4240  IF( LINE .EQ. NIRR ) GO TO 4250
      PRTARY( 1 ) = FORM1
      GO TO 4260
4250  PRTARY( 1 ) = FORM2
4260  DO 4270 J = 1,MXPET
```

```
 4270     PRTARY( J + 1 ) = BLANK
          IF( LINE .EQ. NIRR ) GO TO 4280
          PRTARY( MYSET + 2 ) = FORM4
          PRTARY( MYSET + 3 ) = BLANK
          GO TO 4290
 4280     PRTARY( MYSET + 2 ) = FORM8
          PRTARY( MYSET + 3 ) = FORM6
          GO TO 4290
 4290 CONTINUE
          IF( NIRR .EQ. NMINV ) GO TO 610
C
C    STEP-6
C
C    THE SIZE OF IRREDUNDANT SUBSET IS NOT EQUAL TO THE SIZE OF
C    MINIMUM VECTOR SET.
C
       I = 0
  510  J = I*LSHIFT + LABEL(I+1 )/SCALE
       DO 520 K = 1,NSIRR
          IF( MINV( K ) .LT. 0 ) GO TO 520
          JX = J
          MINX = MINV( K )
          CALL COMPR( JX,MINX,&520 )
          GO TO 540
  520  CONTINUE
       IF( RF - II + 1 .GT.M ) GO TO 530
       DCARX = DCARE( I + 1 ) / RSHIFT
       IF( DCARX/2*2 .NE. DCARX ) GO TO 530
       LABELX = LABEL( I + 1 ) / RSHIFT
       IF( LABELX/2*2 .NE. LABELX ) GO TO 550
       ERROR = .TRUE.
       GO TO 550
  530  LABEL( I + 1 ) = LABEL( I + 1 ) + RSHIFT
       GO TO 550
  540  IF( RF - II + 1 .GT. M ) GO TO 550
       DCARY = DCARE( I + 1 ) / RSHIFT
       IF( DCARY/2*2 .NE. DCARY ) GO TO 550
       LABELX = LABEL( I + 1 ) / RSHIFT
       IF( LABELY/2*2 .EQ. LABELX ) GO TO 550
       ERROR = .TRUE.
  550  I = I + 1
       IF( I .LE. MXVTXL ) GO TO 510
       GO TO 660
C
C    THE SIZE OF IRREDUNDANT SUBSET IS EQUAL TO THE SIZE OF
C    MINIMUM VECTOR SET.
C
  610  I = 0
  620  J = I*LSHIFT + LABEL(I+1 )/SCALE
       IF( FUNC( J + 1 ) .EQ. 2 ) GO TO 630
       IF( RF - II + 1 .GT. M ) GO TO 650
       DCARX = DCARE( I + 1 ) / RSHIFT
       IF( DCARY/2*2 .NE. DCARY ) GO TO 650
       LABELX = LABEL( I + 1 ) / RSHIFT
       IF( LABELY/2*2 .EQ. LABELX ) GO TO 650
       ERROR = .TRUE.
       GO TO 650
  630  IF( RF - II + 1 .GT. M ) GO TO 640
       DCARX = DCARE( I + 1 ) / RSHIFT
       IF( DCARX/2*2 .NE. DCARX ) GO TO 640
       LABELX = LABEL( I + 1 ) / RSHIFT
```

```
      IF( LABELY/2**2 .NE. LABELX ) GO TO 650
      ERROR = .TRUE.
      GO TO 650
  640 LABEL( I + 1 ) = LABEL( I + 1 ) + RSHIFT
  650 I = I + 1
      IF( I .LE. MXVTXL ) GO TO 620
  660 IF( ERROR ) GO TO 670
      GO TO 690
  670 PRINT 680,II
  680 FORMAT( '0','ERROR IN CONSTRUCTING MOS GATE',I3 )
      CALL PRINT( LABEL,NVTEXL )
      RETURN
  690 RETURN
      END
C**********************
C                     *
C   SUBROUTINE MPF    *
C                     *
C**********************
      SUBROUTINE MPF
C   THIS SUBROUTINE OBTAINS MAXIMUM PERMISSIBLE FUNCTION FROM MINIMUM
C   LABEL AND MAXIMUM LABEL AND STORES THE RESULT TO FUNC( NVTEXF ).
      IMPLICIT INTEGER( A-Z )
      INTEGER*2  LABEL  ,DCARE  ,MNL    ,MXL   ,CHAIN  ,FUNC  ,LINK
     1          ,MINV
      COMMON     II     ,N        ,M       ,RF              ,FL
     1  ,LABEL(1024)    ,DCARE(1024)     ,MNL(1024)       ,MXL(1024)
     2  ,CHAIN(1024)    ,STARTL(11)      ,ENDL(11)        ,FUNC(4096)
     3  ,LINK(4096)     ,MINV(4096)      ,STARTF(15)      ,ENDF(15)
      MXVTXL = 2**N - 1
      NVTEXF = 2**( N + II - 1 )
      RSHIFT = 2**( RF - II )
      LSHIFT = 2**( II - 1 )
      DO 10 I = 1,NVTEXF
   10 FUNC( I ) = 0
      I = 0
   20 MNLX = MNL( I + 1 ) / RSHIFT
      MXLX = MXL( I + 1 ) / RSHIFT
      J = I*LSHIFT + MNLX/2
      IF( MNLX/2**2 .EQ. MNLY ) GO TO 30
      IF( MXLX/2**2 .EQ. MXLX ) GO TO 40
      FUNC( J + 1 ) = 2
      GO TO 40
   30 IF( MYLY/2**2 .NE. MXLX ) GO TO 40
      FUNC( J + 1 ) = 1
   40 I = I + 1
      IF( I.LE.MXVTXL ) GO TO 20
      RETURN
      END
C**********************
C                     *
C   SUBROUTINE COMPP  *
C                     *
C**********************
      SUBROUTINE COMPP( JX,MINX,* )
C   THIS SUBROUTINE COMPARES THE SIZE OF TWO VECTORS JX AND MINX
   10 IF( MINX/2**2 .EQ. MINX ) GO TO 20
      IF( JX/2**2 .EQ. JX )RETURN1
   20 JX = JX / 2
      MINX = MINX / 2
      IF( JX.NE.0 .OR. MINX.NE.0 ) GO TO 10
```

```
      RETURN
      END
C*************************
C                        *
C   SUBROUTINE NCUBE     *
C                        *
C*************************
      SUBROUTINE NCUBE
C   THIS SUBROUTINE CONSTRUCTS NCUBE ACCORDING TO THE NUMBER OF EXTERNAL
C   VARIABLES. CHECK THE INPUT VECTOR IN BINARY FORM ASSIGNED TO EACH
C   VERTEX ONE BY ONE AND LINKS THE VERTICES WITH THE SAME WEIGHT.
      IMPLICIT INTEGER( A-Z )
      INTEGER*2  LABEL  ,DCARE  ,MNL    ,MXL   ,CHAIN  ,FUNC  ,LINK
     1         ,MINV
      COMMON     II     ,N       ,M      ,RF              ,FL
     1  ,LABEL(1024)    ,DCARE(1024)    ,MNL(1024)       ,MXL(1024)
     2  ,CHAIN(1024)    ,STARTL(11)     ,ENDL(11)        ,FUNC(4096)
     3  ,LINK(4096)     ,MINV(4096)     ,STARTF(15)      ,ENDF(15)
      MXVTXL = 2**N - 1
      L = N + 1
      DO 10 I = 1,L
   10 STARTL( I ) = 0
      I = 0
   20 X = I
      W = 0
   30 IF( X.EQ.0 ) GO TO 50
      IF( X/2*2.EQ.X ) GO TO 40
      W = W + 1
   40 X = X/2
      GO TO 30
   50 IF( STARTL( W + 1 ).EQ.0 ) GO TO 60
      CHAIN( ENDL( W + 1 ) ) = I + 1
      ENDL( W + 1 ) = I + 1
      GO TO 70
   60 STARTL( W + 1 ) = I + 1
      ENDL( W + 1 ) = I + 1
   70 I = I + 1
      IF( I.LE.MXVTXL ) GO TO 20
      RETURN
      END
C*************************
C                        *
C   SUBROUTINE INPUT     *
C                        *
C*************************
      SUBROUTINE INPUT( CS,* )
C   THIS SUBROUTINE READS DATA INTO LABEL( 2**N ) AND DCARE( 2**N )
C   ON INPUT ERROR, THE FOLLOWING MESSAGE IS PRINTED OUT AND EXECUTION
C   IS TERMINATED.
C        I = FUNCTION NO.  J = CARD NO.  K = COLUMN NO.
      IMPLICIT INTEGER( A-Z )
      INTEGER*2  LABEL  ,DCARE  ,MNL    ,MXL   ,CHAIN  ,FUNC  ,LINK
     1         ,MINV
      COMMON     II     ,N       ,M      ,RF              ,FL
     1  ,LABEL(1024)    ,DCARE(1024)    ,MNL(1024)       ,MXL(1024)
     2  ,CHAIN(1024)    ,STARTL(11)     ,ENDL(11)        ,FUNC(4096)
     3  ,LINK(4096)     ,MINV(4096)     ,STARTF(15)      ,ENDF(15)
      LOGICAL CS
      INTEGER*2 CHAR( 80 )
      INTEGER*2 BLANK/' '/,ZERO/'0'/,ONE/'1'/,DCR/'*'/
C   CHAR( 80 )       CHARACTER BUFFER FOR ONE CARD.
```

```
C    CS                IS SET IF THE GIVEN FUNCTIONS ARE COMPLETELY
C                      SPECIFIED.
       CS = .TRUE.
       IF( 2**N/80*80 .EQ. 2**N ) GO TO 10
       NCARD = 2**N/80 + 1
       GO TO 20
    10 NCARD = 2**N/80
    20 DO 140 I = 1,M
          PRINT 30,I
    30    FORMAT( '0','FUNCTION',I3 )
          VTEX = 1
          DO 100 J = 1,NCARD
             READ 40,CHAR
    40       FORMAT( 80A1 )
             DO 80 K = 1,80
                IF( CHAR( K ) .EQ. BLANK ) GO TO 110
                IF( CHAR( K ) .EQ. ZERO ) GO TO 70
                IF( CHAR( K ) .EQ. ONE ) GO TO 50
                IF( CHAR( K ) .EQ. DCR ) GO TO 60
                GO TO 150
    50          LABEL( VTEX ) = LABEL( VTEX ) + 2**( M - I )
                GO TO 70
    60          DCARE( VTEX ) = DCARE( VTEX ) + 2**( M - I )
                CS = .FALSE.
    70          VTEX = VTEX + 1
    80       CONTINUE
             PRINT 90,CHAR
    90       FORMAT( ' ',3X,80A1 )
   100    CONTINUE
          GO TO 120
   110    IF( J .NE. NCARD ) GO TO 150
          IF( VTEX .NE. 2**N + 1 ) GO TO 150
   120    PRINT 130,CHAR
   130    FORMAT( ' ',80A1 )
   140 CONTINUE
       GO TO 170
   150 PRINT 160,I,J,K
   160 FORMAT( '0','INPUT ERROR IN DATA CARD',3X,'I =',I3,'J =',I3,
      1        'K =',I3 )
       RETURN1
   170 RETURN
       END
C************************
C                       *
C   SUBROUTINE PRINT    *
C                       *
C************************
C   THIS SUBROUTINE PRINTS THE CONTENTS OF AN ARRAY( LABEL,DCARE,ETC. ).
C   ARAY IS THE ARRAY TO BE PRINTED.
C   SIZE INDICATES THE NUMBER OF THE ELEMENTS IN THE ARRAY TO BE
C   PRINTED.
       SUBROUTINE PRINT( ARAY,SIZE )
       IMPLICIT INTEGER( A-Z )
       INTEGER*2  ARAY( 1024 ),PBUF( 16 )
       PRINT 10
    10 FORMAT( '0','*****DUMP*****' )
       DO 60 I = 1,SIZE
       ARAYX = ARAY( I )
       J = 16
    20 IF( ARAYX/2*2 .EQ. ARAYX ) GO TO 30
       PBUF( J ) = 1
```

```
      GO TO 40
30    PBUF ( J ) = 0
40    ARAYY = ARAYY / 2
      J = J - 1
      IF( J.GE.1 ) GO TO 20
      PRINT 50,( PBUF( JJ ),JJ = 1,16 )
50    FORMAT( ' ',16I1 )
60    CONTINUE
      RETURN
      END
```

OGRAPHIC DATA T	1. Report No. UIUCDCS-R-77-896	2.	3. Recipient's Accession No.
e and Subtitle			5. Report Date
IGN OF IRREDUNDANT MULTIPLE-LEVEL MOS NETWORKS FOR TIPLE-OUTPUT AND INCOMPLETELY SPECIFIED FUNCTIONS			September 1977
			6.
hor(s)	CHI-CHUNG YEH		8. Performing Organization Rept. No. UIUCDCS-R-77-896
orming Organization Name and Address			10. Project/Task/Work Unit No.
partment of Computer Science versity of Illinois bana, Ill. 61801			11. Contract/Grant No. NSF No. MCS77-09744
nsoring Organization Name and Address			13. Type of Report & Period Covered
tional Science Foundation 00 G Street, N.W. shington, D.C. 20550			Technical
			14.

plementary Notes

stracts This paper describes a program package for the design of irredundant single-ut or multiple-output MOS networks. This program is an implementation of H.C.Lai's rithms for the design of irredundant MOS networks with a minimum number of MOS s for any given switching function. The given function can be completely specified ncompletely specified and only uncomplemented external variables are permitted as ork inputs. For multiple-output functions, both the case that the outputs are ricted at the last level and the case that the outputs are restricted at the last vels are included, where m is the number of given functions. The output of the ram is a graphic representation of the MOS network configurations. The restric-s on the problem size are described and the listing of the FORTRAN program is uded in the paper.

y Words and Document Analysis. 17a. Descriptors

c design, logic circuits, logic elements, programs (computers).

entifiers/Open-Ended Terms

uter-aided-design, MOS networks, irredundant networks, single-output networks, iple-output networks, optimal networks, MOS cells, MOS, FET.

OSATI Field/Group

ilability Statement	19. Security Class (This Report)	21. No. of Pages
UNLIMITED	UNCLASSIFIED	126
	20. Security Class (This Page) UNCLASSIFIED	22. Price